A Network Management Tool
for the
University of Tennessee at Chattanooga
Using
Simple Network Management Protocol

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee at Chattanooga

Ghasem Shojaie

August 1993

DEDICATION

This thesis is dedicated to my parents

Mr. Seifour Shojaie

&

Mrs. Safiyeh Shojaie

who have continuously given me support and guidance.

ACKNOWLEDGMENTS

I would like to thank Dr. Fred Nixon for his help, and encouragement, both with this thesis and with the courses I have taken from him. I would like to thank Dr. Jack Thompson, Dr. Joseph Kizza and Dr. Terry Walters for serving on my committee, and Mr. Art Hagood for his encouragement, help, and understanding during this thesis.

ABSTRACT

This research is an attempt to develop a general purpose network management tool, using the Simple Network Management Protocol, for the University of Tennessee at Chattanooga (UTC). This tool is intended to be a non vendor specific application that runs on very low cost hardware, yet provides the primary functions one needs to monitor the health of network devices such as routers and bridges. The Simple Network Management Protocol was chosen because it is an industry standard. It has fewer operators than other network management protocols; therefore, it is simpler to implement. Because of the type of network that is used at UTC, the Ethernet interface was chosen for testing the application. As a result of the research, an analysis was done on the Simple Network Management Protocol, and a set of network I/O routines was developed which work with any Ethernet interface for a PC. The I/O functions were written in Borland C++ version 3.0 for PCs. A few I/O routines were adapted from previous research for interfacing with the Clarkson packet driver. The final package was submitted to the Center of Excellence for Computer Application (CECA) to be used as a network management tool at the UTC campus.

Table of Contents

List of Tables

List of Figures

I am submitting a thesis written by Ghasem Shojaie entitled "A Network Management Tool for the University of Tennessee at Chattanooga Using Simple Network Management Protocol." I have examined the final copy of this thesis and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a concentration in computer science.

[redacted]

Dr. John F. Nixon, Major Professor

We have read this thesis
and recommend its acceptance:

[redacted]

Accepted for the Graduate Division:

[redacted]

Director of Graduate Studies

Chapter 1

Introduction

Purpose and Goals

The primary goal of this research is to develop a network management tool to be used at the campus of the University of Tennessee at Chattanooga. The development of the tool required a decision to be made on the protocol to be used for network management. The Simple Network Management protocol (SNMP) was chosen. The program is called SNMPMGR (Simple Network Management for the network Manager). This tool is a vendor independent application (SNMP agent monitor) that runs on very low cost hardware, yet provides the functions one needs to monitor the health of network devices such as routers, gateways, and bridges. SNMP was chosen because it is widely supported by the vendors of devices utilizing the Internet Protocol (IP), and it is recognized as a standard network management protocol by the Internet Community. A description of SNMP is given in the first four chapters of this thesis. Chapter 5 describes the implementation of the tool and its functionalities (SNMPMGR functionality covers all areas of network management listed in Chapter 4). As a result, Chapter 6 presents a list of problems with SNMP and suggested solutions. Appendix D contains a user's guide to the SNMPMGR program.

Network Management Protocol

Because of the importance of a functioning data network, one or more computer system experts, called network engineers, usually are made responsible for installing, maintaining, and troubleshooting the network. For the network engineer, the solution to a network problem could be simple, such as answering a confused user's question, or more complicated, such as replacing failed or malfunctioning equipment or initiating disaster recovery due to a catastrophic event.

Further, as a network expands, the size and number of potential problems also increase, as does the scope and complexity of the network engineer's job. To accomplish their task, engineers must have a great deal of knowledge about the data network. The sheer volume of this information can quickly become unmanageable, particularly as the network grows and changes. To assist the engineers, various network management tools have been developed. It would also be cost effective to have a system that could look out for itself for the most part and, in the process, perform routine tasks (such as collecting the number of errors in fragmenting or reassembling packets) for the engineers.

Network management is the process of controlling a complex data network to maximize its efficiency and the productivity of its users. Depending on the capabilities of the system that administers network management, the network management process will usually include collecting data (such as collecting the number of input errors in an interface), processing data (archiving the number of errors), and then presenting it to the engineers for use in evaluating the operating condition of the network. It may also involve analyzing the data and offering solutions and possibly even handling a situation without involving an engineer [7]. To accomplish all of this, ISO defines the following five functional areas for network management are described in Chapter 4:

1. Fault management
2. Configuration management
3. Security management
4. Performance management
5. Accounting management

The Internet Activities Board (IAB), which oversees the activities in networking technology and protocols for TCP/IP internetworking community, took on the task of coordinating work in selecting a standard network management protocol. Simple

Network Management Protocol (SNMP) was chosen as one of the standard protocols for network management and is based on the Management Information Base (MIB) is a database of managed objects accessed by network management protocols. SNMP is an application protocol, so a standard communication platform must be used to enable SNMP messages (protocol data units) to transfer concurrently with the user data message. To achieve this, SNMP uses the TCP/IP protocol model.

Request For Comments (RFCs) are documents communicating ideas for development in the Internet community which may become Internet standards. RFC 1157 describes the agent/manager station model used in SNMP. "An SNMP agent is software capable of answering valid queries from an SNMP manager station" [11]. The network device that provides information about the MIB to the manager station will have an SNMP agent. For the agent/manager station model to work, the agent must speak the same language as the manager.

At this time, SNMP is the only industry standard for network management protocol. There is a protocol called LAN Man Management Protocol (LMMP) which attempts to provide a network management solution for LAN environments. LMMP was formerly known as Common Management Information Services and Protocol. This protocol was developed by 3Com and IBM. The only devices on the market utilizing this protocol are 3Com and IBM products. In contrast, Simple Network Management Protocol (SNMP) is an industry standard, and all compliant products are required to support the SNMP protocol with their devices. SNMP was chosen for this project because it can be used to manage any device via an SNMP agent that meets the networking standard [7]. As an example, the following vendors products support SNMP: Cabletron, Cisco System, Thomas Conrad, AT&T, Chipcom, Digital Equipment Corp. (DEC), and IBM. Also, SNMP will be the only standard for network management until Common Management Information Protocol (CMIP) is fully defined.

History of SNMP

Before the IAB postulated its initial strategy for network management, a group of four engineers produced a protocol called the Simple Gateway Monitoring Protocol (SGMP) [1] in order to meet the requirements of their line-management responsibilities. As reported in the initial IAB strategy for network management, SGMP was modified to reflect the experience gained by its use and to achieve conformance with the MIB (RFC 1157). The resulting protocol is termed the Simple Network Management Protocol. The IAB emphasized that further development of SNMP should keep the protocol simple and focused on the area of fault management and configuration management. This was the IAB's short-term plan for network management protocol. For the long-term, "the IAB recommended the Internet research community explore the emerging OSI network management protocol" [7], the Common Management Information Protocol (CMIP).

In order to provide for an orderly transition between the short term and long term technologies, it was essential that a framework be developed that was suitable for use with both protocols. To achieve this independence, a set of rules was developed for defining objects which are managed in a network's device (agent). The rules were constructed in such a way as to be independent of the actual management information [2]. These rules are defined in RFC 1065 the Structure of Management Information (SMI) and the RFC 1066 Management Information Base (MIB), which will be explained in the next chapter.

Chapter 2

Management Information Base (MIB)

The MIB is a precise definition of the information accessible via a network management protocol. The MIB uses a hierarchical structure to define the network management information available from a device. Each device, to comply with the standard network management protocol, must use the same structure for storing information that is defined by the MIB.

RFC 1065 describes the syntax and type of information available in the MIB for the management of IP networks. Entitled "Structure Management Information" (SMI), this RFC defines simple rules for naming and creating types of information. Examples of the types of information (objects) allowed by the SMI include a *Gauge,* an integer that may increase or decrease, and *TimeTicks,* an integer which counts time in hundredths of seconds[1]. Before continuing, it is necessary to introduce a typographical convention used in this thesis. The type used throughout the thesis changes in a few places (change to italic) to indicate the name of a variable in the program or in the input of the program (objects configuration file).

Using the rules of the SMI, RFC 1066 presented the first version of the MIB for use with the TCP/IP protocol suite. This standard is called the MIB-I, and it explains and defines the exact information base needed for monitoring and controlling TCP/IP based internets. RFC 1158 proposed a second MIB, MIB-II, for use with the TCP/IP protocol suite. This proposal extends the information base defined in MIB-I by expanding the set of objects defined in the MIB [10]. To facilitate the migration of vendor-specific protocols to a standard management protocol, RFC 1156 allows for expansion of the MIB for vendor specific enhancements. Therefore, companies could add their own objects to the device for further enhancement [6].

Objects in the standard Internet MIB

A summary of the objects most often used in MIB-I and MIB-II for System, Interface, Address Translation, IP, and SNMP groups is shown in the following sections. The objects for the other groups such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) can be found in RFC 1158 [10].

System Group: The system group must be implemented by all managed nodes and contains generic configuration information:

Table 1: System Group Objects

Object	Description
sysDescr	description of device
sysObjectID	identity of the agent software
sysUpTime	how long ago the agent started
sysContact	name of contact person
sysName	device name
sysLocation	device physical location
sysServices	services offered by device

The **sysServices** object is a concise means of determining the set of services which the device potentially offers. It is an integer value that indicates if the device is a repeater or bridge or supports IP, TCP, or SMTP. A complete description of each remaining object can be found in RFC 1158.

Interface Group: The interface group must be implemented by all managed nodes, and contains generic information on the interface. This group contains two top level objects: "the number of interface attachments on the node, and a single table containing information on those interfaces" [10].

ifNumber : the number of interfaces attached.

For each interface there is a table with the following objects in the table.

Table 2: Interface Group Objects

Object	Description
ifIndex	interface number
ifDescr	description of the interface
ifType	type of interface
ifMtu	maximum transmission unit
ifSpeed	transmission rate in bits/second
ifPhysAddress	media specific address
ifAdminStatus	desired interface state
ifOperStatus	current interface state
ifLastChange	how long ago interface changed state
ifInOctets	total octets received from the media
ifInUcastPkts	unicast packets delivered above
ifInNUcastPkts	broadcast/multicast packets delivered above
ifInDiscards	packets discarded due to resource limitations
ifInErrors	packets discarded due to format error
ifInUnknownProtos	packets discarded for unknown protocols
ifOutOctets	total octets sent on the media

Table 2: Interface Group Objects (continued)

Object	Description
ifOutUcastPkts	unicast packets from above
ifOutNUcastPkets	broadcast/multicast packets from above
ifOutDiscards	packets discarded due to resource limitations
ifOutErrors	packets discarded due to error
ifOutQlen	packet size of output queue
ifSpecific	MIB specific pointer

All of these objects are generic in that they apply to all interfaces regardless of the interface type. Support for interface type specific objects is possible by using the **ifSpecific** object as a pointer to some interface specific object. The **ifAdminStatus** object is a means for conveying to the agent an imperative action. For example, if the value is changed from **down** to **up** the agent understands this to mean that the interface should be initialized and brought into the ready state. A complete description of every object can be found in RFC 1158 [10].

Address Translation Group : The address translation group must be implemented by all managed nodes, and contains address resolution information. In fact, the group contains a single table used for mapping IP addresses into media specific addresses[5].

Table 3. Address Translation Group Objects

Object	Description
atIfIndex	interface number
arPhysAddress	media address of mapping
atNetAddress	IP address of mapping

IP Group: The IP group must be implemented by all managed nodes. The group contains several scalars and four tables. These objects define the status of the agent dealing with the Internet Protocol packets.

Table 4.1 : IP Group Scalar Objects

Object	Description
ipForwarding	acting as a gateway or a host
ipDefaultTTL	default TTL for IP packets
ipInReceives	total datagrams from below
ipInHdrErrors	datagrams discarded due to format error
ipInAddrErrors	datagrams discarded due to misdelivery
ipForwDatagrams	datagrams forwarded
ipInUnknownProtos	datagrams destined for unknown protocols
ipInDiscards	datagrams discarded due to resource limitations
ipInDelivers	datagrams delivered above
ipOutRequests	datagrams from above
ipOutNoRoutes	datagrams discarded due to no route
ipReasmTimeout	time-out value for reassemble queue
ipReasmReqds	fragments received needing reassembly
ipReasmOKs	datagrams successfully reassembled
ipReasmFails	reassembly failures
ipFragOKs	datagrams successfully fragmented
ipFragFails	datagrams needing fragmentation but the IP flags field said not to
ipFragCreates	fragments created

The IP address table (**ipAddrTable** an object in IP group that represents the IP address table) keeps track of IP addresses associated with the managed node, and each row of the table contains the following objects.

Table 4.2 : IP Address Table

Object	Description
ipAdEntAddr	the IP address of this entry
ipAdEntIfIndex	interface number
ipAdEntNetMask	subnet mask for IP address
ipAdEntBcastAddr	LSB of IP broadcast address
ipAdEntReasmMaxSize	the largest IP datagram able to be reassembled

IP routing table objects and IP address translation table objects can be found in RFC 1158. These objects define the status of the agent with the routing table and the destination address translation.

SNMP Group: A set of managed objects that were defined for SNMP applications in MIB-II is shown in the following table.

Table 5. SNMP Group Objects

Object	Description
snmpInTooBigs	PDUs received with error status of tooBig
snmpInNoSuchNames	PDUs received with error status of noSuchName
snmpInBadValues	PDUs received with error status of badValue
snmpInReadOnlys	PDUs received with error status of readOnly
snmpInGenErrs	PDUs received with error status of genErr
snmpInTotalReqVars	number of MIB objects retrieved
snmpInTotalSetVars	number of MIB objects changed
snmpOutTooBigs	PDUs sent with error status of tooBig
snmpOutNoSuchName	PDUs sent with error status of noSuchName
snmpOutBadValues	PDUs sent with error status of badValue
snmpOutReadOnlys	PDUs sent with error status of readOnly
snmpOutGenErrs	PDUs sent with error status of genErr
snmpEnableAuthTraps	enable/disable the generation of authenticationFailure traps

More information about the MIB defined object and group can be found in RFC 1158. The next chapter introduces the SNMP itself.

Chapter 3

SNMP Protocol

Only four primitive operations are available in the protocol:

- **get,** which is used to retrieve specific management information from an agent's MIB.
- **get-next,** which is used to retrieve sequential traversal of management information (object and tables defined in a MIB).
- **set,** which is used to modify management information in an agent's MIB.
- **trap,** which is used to report extraordinary events [2].

The SNMP agents and manager stations communicate by sending messages. Each of these messages are single packet exchanges. Because of this, SNMP uses UDP (User Datagram Protocol) as the layer 4, or transport layer, protocol. UDP is a connectionless service, so SNMP does not to have to maintain a connection between an agent and manager station to transmit messages. Figure 1 shows the ISO Reference Model for SNMP. A brief description of each layer is given in the following page.

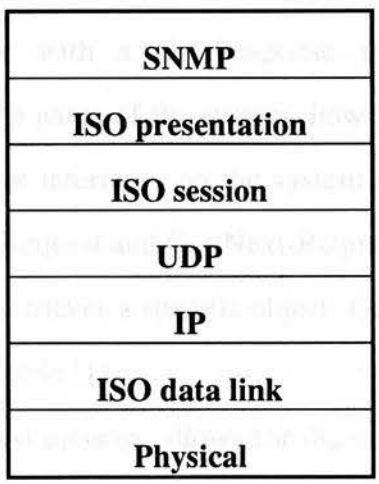


Figure 1. SNMP in the ISO Reference Model

The Physical and data link layers are independent of SNMP, but the next two layers are specified by SNMP (UDP for the Transport layer and IP for the Network layer). The Presentation layer and Session layer can be neglected if the ISO model is not being used. The top most layer is the Application layer which is the SNMP application.

SNMP is an asynchronous request/response protocol. This means that an SNMP entity need not wait for a response after sending a message. It can send other messages or do other activities. Further, since the request or response might be lost by the underlying transport service, it is up to the sending SNMP entity to implement the desired level of reliability. SNMP uses five types of standard messages to implement the above operations [2]:

1. Get-Request
2. Get-Response
3. Get-Next-Request
4. Set-Request
5. Trap

The SNMP manager station uses the Get-Request message to retrieve information from a network device that has an SNMP agent. The SNMP agent in turn responds to the Get-Request message with a Get-Response message. Information that might be exchanged includes the name of the system, how long the system has been running, and the number of network interfaces on the system according to the object defined in the MIB (Table 4). Get-Request and Get-Next-Request used in conjunction obtain a table of objects. Get-Request retrieves a specific object; Get-Next-Request then is used to ask for the next object in the table [1].

The Set-Request message allows for the remote configuration of parameters on a device. Examples of Set-Request messages include setting the name of a device, shutting an interface down administratively, or clearing an address resolution protocol table entry.

The SNMP Trap message is an unsolicited message an SNMP agent sends to a

manager station. These messages inform the manager station about the occurrence of a specific event. For example, SNMP Trap messages can be used to inform the network management system that a circuit has just failed, the disk space of a device is nearing capacity, or a user has just logged onto a host. Seven types of SNMP traps are defined as part of MIB-II as follows [11]:

1. Coldstart of a system
2. Warmstart of a system
3. Link down
4. Link up
5. Failure of authentication
6. Exterior Gateway Protocol (EGP) neighbor loss
7. Enterprise-specific

The Coldstart trap message indicates that the agent sending the trap is reinitializing itself such that its configuration or protocol implementation has changed. A Coldstart trap occurs when a device is powered on. The Warmstart trap message indicates that the device sending the trap is reinitializing itself such that its configuration or protocol implementation has not changed. If the SNMP agent in a device is reset, this action would invoke a Warmstart trap. Typically, this event occurs because of manual intervention.

The Link Down trap means a specific link on the source device has failed. A Link Up trap signifies a specific link from the source device has become available. A Failure of Authentication trap message is sent to the network management system if an SNMP agent determines that a request does not provide proper authentication, for instance, if the agent gives the wrong SNMP community string (community strings are explained later in this section).

An Exterior Gateway Protocol (EGP) neighbor loss trap message is used by an SNMP agent to report the loss of an EGP neighbor. EGP is a reachability protocol used between data networks. Enterprise-specific traps are used by the private enterprise specific messages.

Community Strings

"The SNMP protocol does not provide information or allow configuration changes of a network device without some kind of security" [7]. The SNMP agent in the network device requires the SNMP manager to send a particular password with each message. This password is referred to as the SNMP community string. The SNMP agent then can verify if the station is authorized to access the MIB information. It is possible for a SNMP agent to allow different levels of security using a community string. For example, the agent could define a community string to allow only Get-Request and Get-Next-Request messages. This would allow read-only access to the information in the MIB. The agent could allow Get-Request, Get-Next-Request, and Set-Request messages using a different string that would allow full (read-write) access to the agent.

"Community strings are within SNMP packets in clear ASCII text" [7]. With little effort, a network literate person can learn the community string used by a given SNMP agent. This easy access presents a security problem for SNMP.

Data Encoding

Although an encoding may be complex overall, the actual rules used to produce the encoding are small in number and quite simple to describe. The Basic Encoding Rules (BER) are recursive algorithms that can produce a compact octet encoding for any ASN.1 (Abstract Syntax Notation One) value [11]. The Abstract Syntax Notation (ASN) is the language OSI uses to define the format of Protocol Data Units (PDUs) exchanged by an arbitrary protocol. It is also used by SNMP for defining the objects which are managed in an agent.

At the top-level, the BER describes how to encode a single ASN.1 type. This may be a simple type such as an INTEGER, or an arbitrarily complex type. Conceptually, the key to applying the BER is to understand that the most complex ASN.1 type is nothing more than a number of smaller, less complex ASN.1 types. If this decomposition continues, then ultimately an ASN.1 simple type (such as INTEGER) is encoded. Using the BER, each ASN.1 type is encoded as three fields:

- a tag field, which indicates the ASN.1 type;
- a length field, which indicates the size of the ASN.1 value encoding which follows;
- a value field, which is the ASN.1 value encoding;

Figure 2 shows the three fields used in ASN.1.

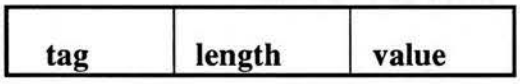


Figure 2. ASN.1 Fields

Tag Field

The tag field is encoded as one or more identifier octets. This encoding must somehow capture the definition of the corresponding ASN.1 type. The BER does this by

encoding the ASN.1 tag of the type. There is a tag associated with each type defined using ASN.1. There are four classes of tags in ASN.1:

- Universal tags, for the well known data types.
- Application-wide tags, which are defined within a single ASN.1 module.
- Context-specific tags, which are used to provide distinguishing information in constructor types.
- Private-use tags, which are used by consenting parties [11].

In addition to belonging to one of these classes, a tag has associated with it a non-negative integer. Thus, the tag field, officially termed the identifier octets, generated by the BER must encode not only the tag's class but also the tag's number.

The tag field must encode one other field. An ASN.1 type might be primitive (such as integer) or it might be constructed (such as structure), and this information is encoded in the f field. A value of 0 indicates a primitive type, and a value of 1 indicates a constructed type.

Figure 3. shows the order of the tag field.

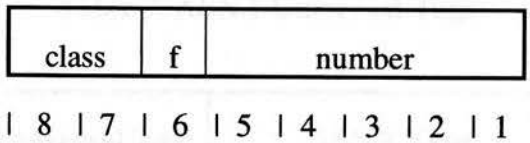


Figure 3. Tag Field

The encoding of the class field (bit 7 and 8) are shown in Table 6.

Table 6. Class for the tag field

Class	Bit 8	Bit 7
Universal	0	0
Application-wide	0	1
Context-specific	1	0
Private	1	1

Thus, five bits are left over for encoding the non-negative number. If the tag's number is less than 31, then it is encoded in the five bits that remain. Otherwise, bits 5 through 1 are set to all ones, which indicates that the octets that follow contain the tag's number. Therefore, in many cases, and clearly for all the Universal tags defined in table 7, a single octet is sufficient to encode the tag field. In the other cases, one or more octets follow the first octet. The high-order bit (8) indicates whether this particular octet is the last octet of the tag field.

Table 7. ASN.1 Universal Tags

Universal Tag	ASN.1 Type
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
7	ObjectDescriptor

Table 7. ASN.1 Universal Tags (continued)

8	EXTERNAL
9	REAL
10	ENUMERATED
11-15	Reserved for addenda
16	SEQUENCE, SEQUENCE OF
17	SET, SET OF
18	NumericString
19	PrintableString
20	TeletexString
21	VideotexString
22	IA5String
23	UTCTime
24	GeneralizedType
25	GraphicsString
26	VisibleString
27	GeneralString
28	CharacterString
29-...	Reserved for addenda

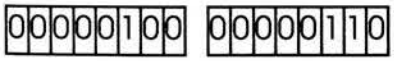
Length Field

The length field is encoded as one or more octets. This encoding indicates how many octets make up the value field of the ASN.1 type being encoded. If the length is longer than one octet, then the high-order bit (8) of the first octet is set to one and the remaining bits indicate the number of octets in the length field. The length is encoded using the unsigned integer representation found by concatenating the specified number of octets that follow the initial length octet.

Value Field

The value field is encoded as zero or more octets. If the object is a simple INTEGER or OCTET STRING the value is simply inserted in the value field. If the value is an OBJECT IDENTIFIER then rules defined in MIB-II for object identifier are to be followed.

The following example shows the encoding of the community string "public" used in SNMPMGR. The class field is set to 00 to indicate Universal tag, the f bit is set to 0 to indicate primitive type such as string, and the number field is set to 4 to indicate the tag's number 4 for **OCTET STRING**. The length field in set to 6 to indicate the length of community string "public".



class f number length

"p" "u" "b" "l" "i" "c"

Chapter 4

SNMP Group Objects in Network Management

The SNMP group of objects are helpful in all five areas of network management [7]. The following sections contain explanations of how SNMP could be used in each of the network management areas.

Fault Management

The SNMP group objects in Table 8 apply to fault management. Each object gives information about errors concerning the packet handling of the agent. RFC 1157 defines each of these errors. While an agent's errors might not indicate a problem with the network itself, it may indicate that an entity is not handling SNMP packets properly [2]. The number and types of errors also can indicate that the entity is receiving SNMP packets with errors from network devices. The solutions to these errors often reside in the configuration of either the SNMP manager or the agent.

Table 8. SNMP Objets for Fault Management

Object	Description
snmpInASNParseErrs	total input ASN.1 errors
snmpInTooBigs	total input 'tooBig' errors
snmpInNoSuchNames	total input 'noSuchName' errors
snmpInBadvalues	total input 'badValue' errors
snmpInReadOnlys	total input 'readOnly' errors
snmpInGenErrs	total input 'genErr' errors
snmpOutTooBigs	total output 'tooBig' errors
snmpOutNoSuchNames	total output 'noSuchName' errors
snmpOutBadValues	total output 'badValue' errors
snmpOutGenErrs	total output 'genErr' errors

Performance Management

The SNMP objects in Table 9 apply to performance management. The **snmpInPkts** and **snmpOutPkts** can determine what percentage of resources an agent is using to handle the SNMP input and output packets [7]. Monitoring the rate of change of these objects can suggest the cause of a high SNMP packet input or output rate. For example, a high rate of **snmpInGetRequests** and **snmpOutGetResponses** may indicate that a manager is currently gathering information from the agent. The following table shows a list of objects that could be used for performance management.

Table 9 : SNMP Objects for Performance Management

Object	Description
snmpInPkts	number of SNMP packets input
snmpOutPkts	number of SNMP packets sent
snmpInTotalReqVars	number of Get/Get-Next Requests input
snmpInTotalSetVars	number of Set-Requests input
snmpInGetRequests	number of Get-Requests input
snmpInGetNexts	number of Get-Next-Requests input
snmpInSetRequests	number of Set-Requests input
snmpInGetResponses	number of Get-Responses input
snmpInTraps	number of Traps input
snmpOutGetRequests	number of Get-Requests output
snmpOutGetNexts	number of Get-Next-Requests output
snmpOutSetRequests	number of Set-Requests output
snmpOutGetResponses	number of Get-Responses output
snmpOutTraps	number of Traps output

Accounting Management

Some of the objects useful for performance management also apply to accounting management, as shown in Table 10. Instead of using these objects to find the rate at which packets enter and leave the agent, as in performance management, accounting management applications can use them to find a total number for each type of SNMP packet sent and received (information that can be useful for network billing based on the number of packets sent or received) [7].

Table 10. SNMP Objects for Accounting Management

Object	Description
snmpInPkts	Total SNMP packets input
snmpOutPkts	Total SNMP packets sent
snmpInTraps	Total Traps input
snmpOutTraps	Total Traps output

Security Management

One aspect of security management involves tracking failed authentication attempts. The actions taken to do this might include checking for unsuccessful password entries for a computer login or for invalid community string in the SNMP messages. **snmpInBadCommunityNames** counts the number of times a user or application, when attempting to communicate with an SNMP on an agent, does not give the correct community string [2]. **snmpInBadCommunityUses** counts the number of times an SNMP packet was received that had a community string that did not allow the requested operation. In many network devices, different community strings can be set up for different operations [2]. For example, one community string might authorize the Get-Request and Get-Next-request operations; another might allow other operations in SNMP agents.

Configuration Management

snmpEnableAuthenTraps is the only object that applies to configuration management. By definition, an agent must have the ability to send an SNMP Authentication Failure trap when it receives a SNMP packet with an incorrect community string. Therefore, this object is used to specify for the destination of the Authentication Failure trap. SNMP does not define a complete set of objects for configuration management. Usually, vendor specific objects are used for configuration of the devices.

Chapter 5

A Network Management Application

(Implementation)

This chapter contains a description of the design and implementation of the network management tool (SNMPMGR). The code for this tool is included in Appendices A and B. There are many network management applications on the market, but they are either vendor specific or require a specific workstation on which to run (such as SUN workstation). The goals of SNMPMGR are to be independent from any specific vendor's product and to not require a specific brand of computer to execute. SNMPMGR was designed to run on a personal computer with an Ethernet interface card. It accesses the predefined MIB objects of network devices and displays their value. Since the application was designed for the University of Tennessee campus network, the Ethernet type of network was chosen for the device interface. The Clarkson packet driver (a shareware packet driver for interfacing with the network) was used to test the application, and a set of routines for interfacing with the packet driver was adapted from previous research in the area. These routines are documented in Appendix B[13].

Program Design

A top-down modular design was used to implement the program. A structure chart was used to depict the decomposition of the program to smaller modules (Figure 4). The modules are designed to be general purpose, so they can be used easily for future development. The size of each module is a maximum of 100 lines of code. A set of general purpose string manipulation routines was developed which was used throughout the program. Since these string functions were used by all the routines, they are not shown in the structure chart.

Figure 4. Program Structure Chart.

The program initially was broken down into three main functions:

1) Initiate Processing: It is implemented by 'initsnmp()' routine.

2) Main Processing: It is implemented by 'pollagents()', 'rcvpkts()', 'dispall()', 'getkey()', 'ckopstat()', and 'beep()' routines.

3) Terminate processing: It is implemented by 'endsnmp()' routine.

A complete program structure chart is shown in Figure 4. The following section describes the main data structures that were used in the implementation process of the program.

Data Structures

The SNMP Database in Memory

The user is required to provide the name of the objects to be displayed in an ASCII text file called objects configuration file. The name of this objects configuration file is provided to the application on the command line. When the application reads the user's objects configuration file (the format of the file is described in the following sections), a database is created in memory. The data from the file are arranged in memory in agent object lists which are linked together in the order they appeared in the file. Each agent's name has associated with it a block of data containing the agent's name (i.e., its Internet address) as a null-terminated string of ASCII characters, its community in a similar ASCII string, and a pointer to a string of object information blocks.

Object information blocks are allocated, one to each object, for an agent. An object information block contains the following information: the object name as a string of ASCII characters terminated by a null character, the object's index (described in the indices section) if any, a pointer to the object's current value (the storage for the value itself is allocated separately), an integer variable called *changflag* indicating if the

object's value has changed since the object was last accessed by the user's application, and a time stamp indicating the last time the value was set.

The *changflag* is set to one by the program every time the agent sends a new value for the object. When the object is displayed the *changflag* is set to zero to indicate the current value of the object was displayed. The object will not be displayed until a new value for the object is received (when the *changflag* is one).

The Management Information Object Structure

Another important data structure is the object information structure. It contains a pointer to a character string (*variable*), a generic pointer which may point to various objects (*nxtblock*), a variable telling what type of object the generic pointer is indicating (*objtype*), a variable for the type of ASN.1 value (integer, counter, string, etc.) the object has associated with it (*vartype*), and a pointer to a function which returns an integer. When SNMPMGR generates its working version of the MIB, each object, whether it has a value associated with it (ifOperStatus) or it is made up of other objects (ifEntry), has an associated object information structure.

MIB Structure Elements

In the memory representation of the MIB, the object name pointer points to the null-terminated ASCII string containing the object's name. The *objectflag* describes the type of object pointed to by the generic pointer. If the *objectflag* is zero, the defined object has a value and the general pointer may be either null or pointing to an array of strings. The strings are ASCII representations of the numerical values of the object. If the value *objectflag* is a one, the pointer in the array with an index of one points to the proper ASCII string. The type field is used to indicate, when applicable, what types of values (integer, string, counter, etc.) can be taken by the MIB object defined by the structure.

The MIB array of structures

The MIB structure is defined as a tree in the Internet Community as shown in Figure 5 (only parts of the tree that are important to SNMP are shown). Each level of the tree is associated with a group of objects defined in Chapters 2-4. The same concept is used to implement the MIB for this application. The memory representation of MIB in this application is simply a mechanism for translating from MIB objects to object identifiers in an SNMP message, and vice versa. It consists of arrays of object information structures linked together in the form of a tree.

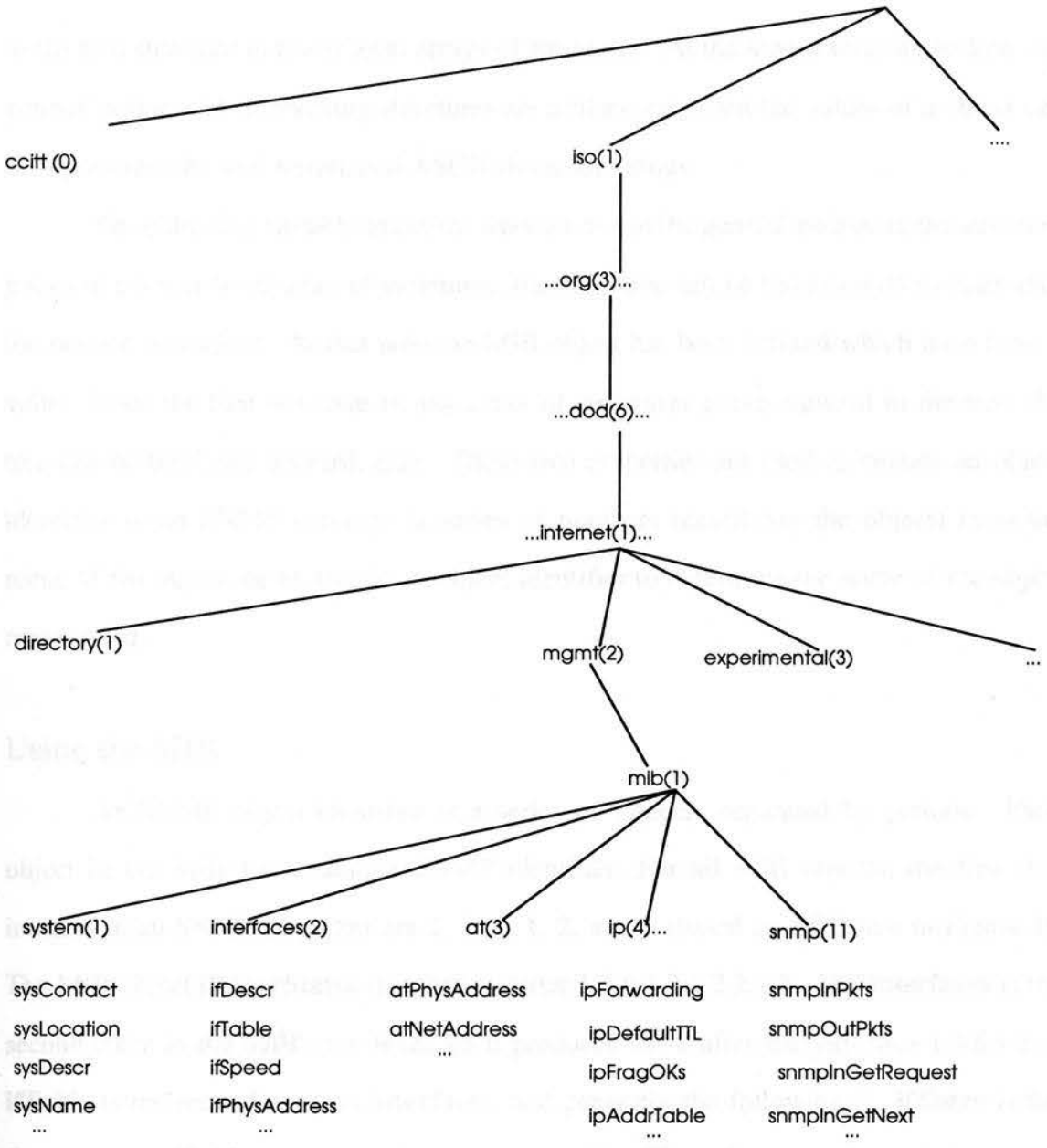


Figure 5. Example MIB Tree Structure

At each level in the tree is an array of structures. There is a structure at the appropriate level in the MIB for each object. The general pointer of the first structure in each array indicates the structure at the next higher level which points to this array. A structure with null pointers and a zero flag object is the last structure in each array. At the higher levels in the tree the general pointers in intervening structures in an array point

to the first structure in lower level arrays of structures. At the lowest level in the tree, the general pointers of intervening structures are null except when the values of a object can be represented by null-terminated ASCII character strings.

Since the flag variable indicates whether or not the general pointer in the structure points to a lower level array of structures, the MIB tree can be traversed downward until the bottom is reached. At that point an MIB object has been defined which must have a value. Since the first structure in any array of structures points upward in the tree, the tree can be traversed upward, also. These two properties are used to encode an object identifier in an SNMP message (a series of numbers identifying the object) from the name of the object, or to decode an object identifier to determine the name of the object represented.

Using the MIB

An SNMP object identifier is a series of integers separated by periods. Each object in the MIB has a unique SNMP identifier. For all MIB objects, the first five integers in an SNMP identifier are 1, 3, 6, 1, 2, and 1 (based on MIB tree in Figure 5). The MIB object **ifOperStatus** has the identifier 1.3.6.1.2.1.2.2.1.8. The **interfaces** is the second entry in the MIB's top level, so it produces the 2 after the sequence 1.3.6.1.2.1. **ifTable** is the second entry in **interfaces**, and generates the following 2. **ifEntry** is the first entry in **ifTable** to account for the next 1. The **ifOperStatus** is the eighth entry in **ifEntry**, for the final 8. This produces the SNMP object identifier for **ifOperStatus** of 1.3.6.1.2.1.2.2.1.8. From the top of the MIB, one must descend through the MIB objects **interfaces**, **ifTable**, and **ifEntry** to get to **ifOperStatus**.

Decoding identifiers

The MIB tree is arranged so that in each array of structures at any level, the array element index for the object of interest at that level corresponds to its equivalent digit in the SNMP identifier. Given that the MIB is identified by 1.3.6.1.2.1, the **interfaces** structure's index is 2 to correspond to the next integer in the identifier, the **ifTable** structure's index is 2 in its array of structures, the **ifEntry** structure's index is 1, and the **ifOperStatus** structure's index is 8. Thus, 1.3.6.1.2.1.2.2.1.8 eventually leads to the **ifOperStatus** object.

Encoding identifiers

Because SNMP identifier encoding rules prevent any integer in an identifier from being zero, the zeroeth element in each structure array is not used to decode an identifier. In the MIB, the general pointer in the structure for the object **ifEntry** points to the array of structures containing the one for **ifOperStatus**. However, it really points to the first element (the one with an index of zero) in the array of structures. Since the general pointer in that structure will never point to an object because its index is zero, it can be used to point "backwards" to the structure for **ifEntry**.

When the general pointer in the first structure in an array of structures is a null pointer the top of the MIB tree has been reached. The MIB prefix of five integers can then be added to the existing list of indices to produce a complete SNMP identifier.

To ease finding a specific object's structure in the MIB, a set of pointers are provided. One pointer is available for each object so its structure can be found without traversing the set of MIB structures. The object information structure pointer 'mibroot' points to the highest level MIB array of structures. Its member structures are for the objects **system, interfaces, at, ip, icmp, tcp, udp,** and **egp**.

SNMP Trap Message Structures

When SNMP agents generate unsolicited information, or trap, messages, the type of message is indicated by an integer in the message. This integer is used as the index in an array of object information structures. The function pointer in the selected structure points to the function which will perform a task appropriate to the type of trap message received. If the object specified in the trap message is one being monitored by the user's program, that object's value will be modified to match that in the trap message. The change flag and time stamp will also be updated. If the object in the trap message is not being monitored, a message is displayed on the console indicating the content of the message.

The Socket Data Structure

To store information related to the types of active network connections while the application is running, a socket data structure is created for each connection. This is external to the SNMP program but it is required for interfacing with the Ethernet connection. A socket is used so that multiple connections from the computer running the application to other devices may be maintained simultaneously. A received packet contains a remote socket number which associates it with a program running on the remote device, and a local socket number which links it with the application running on the local computer. The elements of the socket data structure are 1) a pointer to another socket data structure (so structures can be linked together), 2) a pointer to an agent information structure as previously described, 3) a socket identifier (also called a network descriptor), 4) a request identifier number used for linking a reply message to a request message, 5) a network interface driver option object, 6) the remote computer's Internet address, 7) the remote computer application's socket number, 8) the local socket number, and 9) a network protocol indicator. These structures are linked together using

the socket data structure pointers, with the object 'socketroot' pointing to the beginning of the chain.

Packet Driver Data Structures

The part of the program which communicates with the packet driver for the network interface also uses a number of structures and queues. An Address Resolution Protocol (ARP) table of Internet address and Ethernet address equivalencies is maintained through a chain of structures. Packets received by the packet driver are stored on a packet queue, from which they may be removed and added to either the UDP packet queue or the Transmission Control Protocol queue. Internet Control Message Protocol packets and ARP packets are processed immediately, so they require no queues.

Objects Configuration File Format

To know which objects are associated with which agents, the application must be given a objects configuration file name that contains agent names and object names for the agent. To identify which objects are to be accessed in which agent, the information is separated into groups of objects associated with an agent, called a object list. Groups of object lists associated for an agent are always associated with a community string. To differentiate between community strings, agent names, object names, and indices, the text file must be generated in a specific format. The objects configuration files included in Appendix C were used during the testing process of the tool. Since there is not a standard for defining this objects configuration file, this format was created for this application only.

Community and Object Lists

A community list is a community string, which is followed by one or more object lists. Community lists are terminated by the end of the objects configuration file, or by encountering another community string. Object lists within a community list are separated by blank lines. Lines containing comments are not considered blank lines.

Each object list must be part of a community list. Therefore, the first non-comment and non-blank line must contain a community string. After a community has been specified, that name is used as the default community string for following object lists until a new community string is encountered. When a new community string is encountered, it becomes the default community string. Any object lists not immediately following a community string are given the default community string. Community strings have unique properties so that community lists are separated from each other.

An object list consists of the agent's name followed by the objects and associated indices to be accessed in that agent. The index for an object must be on the same line as

the object and separated from it only by one or more spaces or tab characters. No blank lines may appear in an object list.

Community strings

A community string must start at the beginning of a line, and must not begin with the '#' character ' " ' character. Should it do so, it will be considered a comment, not a community string. No other names in the SNMP database file start at the beginning of a line. A community string is terminated by a space, tab, or end-of line character, unless the name is enclosed in double quote characters. When a community string begins with a double quote character, (i.e., the double quote character is the first character on the line) the community string is terminated by another double quote character and so may include space or tab characters. The double quote characters enclosing a name are not considered part of the name when SNMP messages are generated.

Agent names

Since agent names must not start in the first character position on a line, they must be preceded by one or more spaces or tab characters. If the agent name immediately follows a community string, the agent name may be on the same line as the community string, but separated from it by one or more spaces or tabs. If a community string is not supplied immediately before an agent name, the application will assign the default community string to that agent. An agent name is terminated by a space, tab, or end-of-line character. The agent name can be only the agent's Internet address in standard Internet "dot" notation (such as 192.239.44.1). Since no spaces or tabs are allowed in an agent name, it need not be enclosed in double quotes. If it is enclosed in double quotes, they will be ignored.

Object names

Like agent names, object names may not begin with the first character on a line and are terminated by a space, tab, or end-of line character. Valid object names were defined in Chapters 2-4, and must be entered exactly in the form presented there, including spelling and capitalization. Object names need not be enclosed in double quote characters because no spaces are allowed in them.

Object names must not be separated from each other, or the agent name they are associated with, by blank lines. Blank lines separate object lists from each other.

Indices

An index must be on the same line as the object it modifies. It must be separated from the object name by one or more spaces or tabs. If an object name has no index on the same line, the application assumes no index is to be associated with that object.

An index is specific to its associated object. Some objects do not require indices (such as **ifNumber** or **sysUpTime**); others require multiple indices, such as interface objects. When multiple indices are required, they are separated by either a single space or comma. When they are separated by a space, the entire set of indices must be enclosed by a pair of double quote characters. When double quotes are not used, the index name is terminated by the first space, tab, or end-of-line character encountered.

Comments

A comment is initiated by a '#' character either at the beginning of a line in the objects configuration file, or elsewhere in a line preceded by a space or tab character. A comment is terminated by the end of the line on which it was started. A '#' character not at the beginning of a line and not preceded by a space or tab character will not be considered as the beginning of a comment, but will be included in the field currently being scanned.

Local Configuration information

The only network interface configuration information required to run the application is that for a packet driver. Three objects must be provided: the local interface's IP address, the IP network mask, and the local gateway's IP address. All three values are assigned by the manager of the network on which the host running this application resides.

Local Configuration information is entered in the objects configuration file in a format similar to that of a community list. The community string for the local configuration information must be '<Configuration>' ("<>" brackets included). The agent field must be 'pktdvr'. The objects are 'locipaddr=x.x.x.x' for the local IP address, 'ipmask=x.x.x.x' for the IP network mask, and 'gw=x.x.x.x' for the local gateway's IP address. In the examples each 'x' is an integer between 0 and 255, inclusive. If the network is not using any subnet, the value of ipmask would be 255.255.255.0 to indicate there is not any subnet for the network.

Program Structure and Execution Flow

When the program starts, it reads the supplied objects configuration file and creates the database in memory by calling the routine 'initsnmp()'. This routine initializes the SNMP data structures and creates the memory-resident copy of the SNMP agent and object database. The 'initsnmp()' routine calls 'initmib()' to create the MIB tree. It calls 'initudp()' to initialize the packet driver. Also, it calls 'initrcv()' to initialize the routine to handle the trap messages. Upon the successful return from 'initsnmp()', the routine 'pollagents()' is called to interrogate the specified agents for their values of the indicated objects. The 'pollagents()' routine calls 'getreqid()' to find the object identifier. It calls 'getconnid()' to get network connection identification for Get-request message. Also, it calls 'sendudppkts()' to send the Get-request message to the agent. If the call to 'pollagents()' completes successfully, the memory resident database of agent objects may be accessed and the values tested for any abnormalities. Finally, the values of the objects are printed on the console using 'dispall()'. Then the program goes inside a loop (main loop) to repeat the above process until the exit key is pressed (**CTRL-END**). The top level flow chart of the program and the structure chart of the program is shown in Figure 6 and 4.

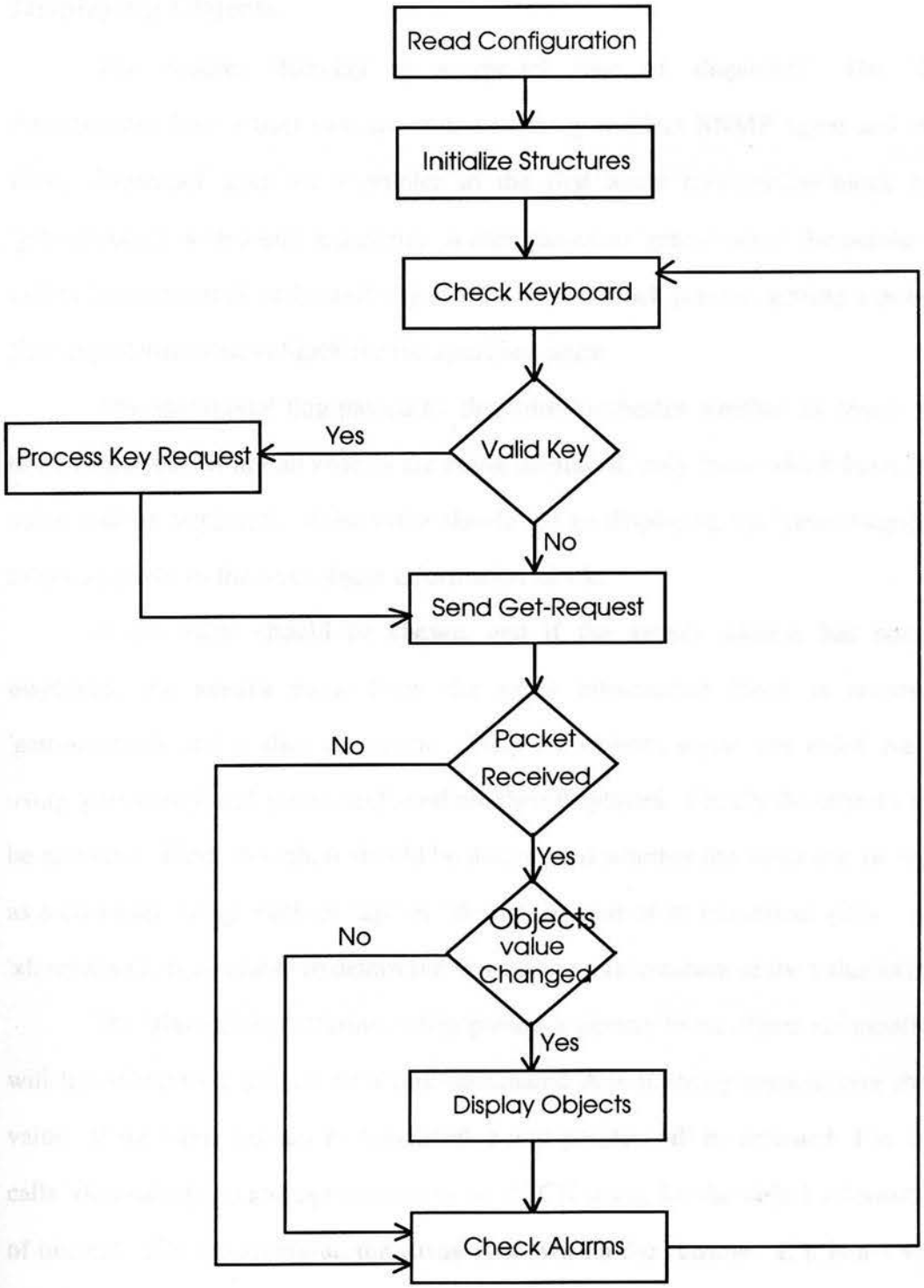


Figure 6. Program Top Level Flow Chart.

Displaying Objects

The routine 'dispall()' is a special case of 'dispinfo()'. The 'dispinfo()' demonstrates how a user can access the memory-resident SNMP agent and object list. First, 'dispinfo()' asks for a pointer to the first agent information block by calling 'getnxtroutp()' with a null argument. It then passes to 'getnxtvarp()' the pointer from the call to 'getnxtroutp()' and a null object information block pointer, getting a pointer to the first object information block for the specified agent.

The operational flag passed to 'dispinfo()' indicates whether all object values are to be displayed. If not all objects are being displayed, only those which have changed in value will be displayed. If the value should not be displayed, the 'getnxtvarp()' is called to get a pointer to the next object information block.

If the value should be shown, and if the agent's address has not yet been displayed, the agent's name from the agent information block is retrieved using 'getroutaddr()' and is then displayed. Then the object's name and index are retrieved using 'getvarstr()' and 'getinxstr()', and are then displayed. Finally the object's value is to be accessed. First, though, it should be determined whether the value can be represented as a character string, such as "up" or "down", instead of its numerical value. A routine, 'xlatevalue()', is available to determine if a string representation of the value exists.

The 'xlatevalue()' routine, when passed a pointer to an object information block, will try to return a pointer to a null-terminated ASCII string representing the object's value. If the value can not be translated, a null pointer will be returned. The 'dispinfo()' calls 'xlatevalue()' to attempt to retrieve an ASCII string for the object information block of interest. If it is successful, the string is shown on the console. If it is not successful, the object's actual value must be displayed. For retrieving the value, the type of object is first determined by calling 'getvartype()'. This returns a pointer to a null-terminated ASCII string of characters which indicates the object type. From there it can be displayed

correctly by calling the 'printf()' function. Finally, the object's time stamp is retrieved using 'getvartime()' and is displayed as shown in Figure 7.

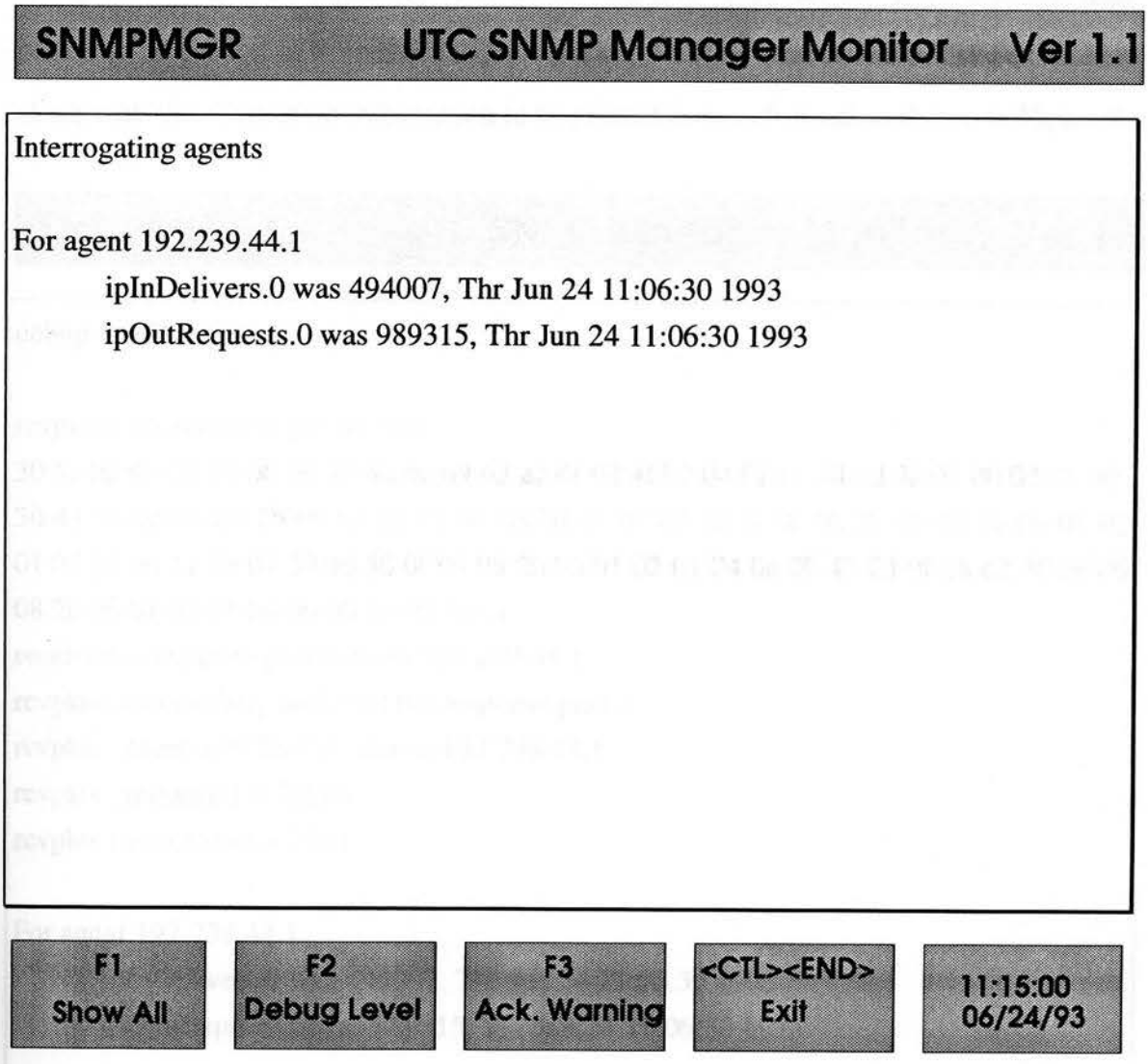


Figure 7. Example Display Screen.

After the time stamp has been displayed, the object's change flag is set to zero and 'getnxtvarp()' is called to find the next object to process. Failing to find another object causes 'getnxtroutp()' to be called to find the next agent information block. When the last object in the last agent information block has been processed, 'dispinfo()' terminates and control is returned to 'main()'.

At the beginning of the main loop the keyboard is tested. If function key 1 (**F1**) has been pressed, 'dispall()' is called. If the function key 2 (**F2**) has been pressed the debug level is entered (either 0, 1, or 2). Debug level 1 causes the content of the next packet to be printed in hexadecimal. Debug level 2 causes the content of the next packet along with the connection information to be printed in hexadecimal as shown in Figure 8.

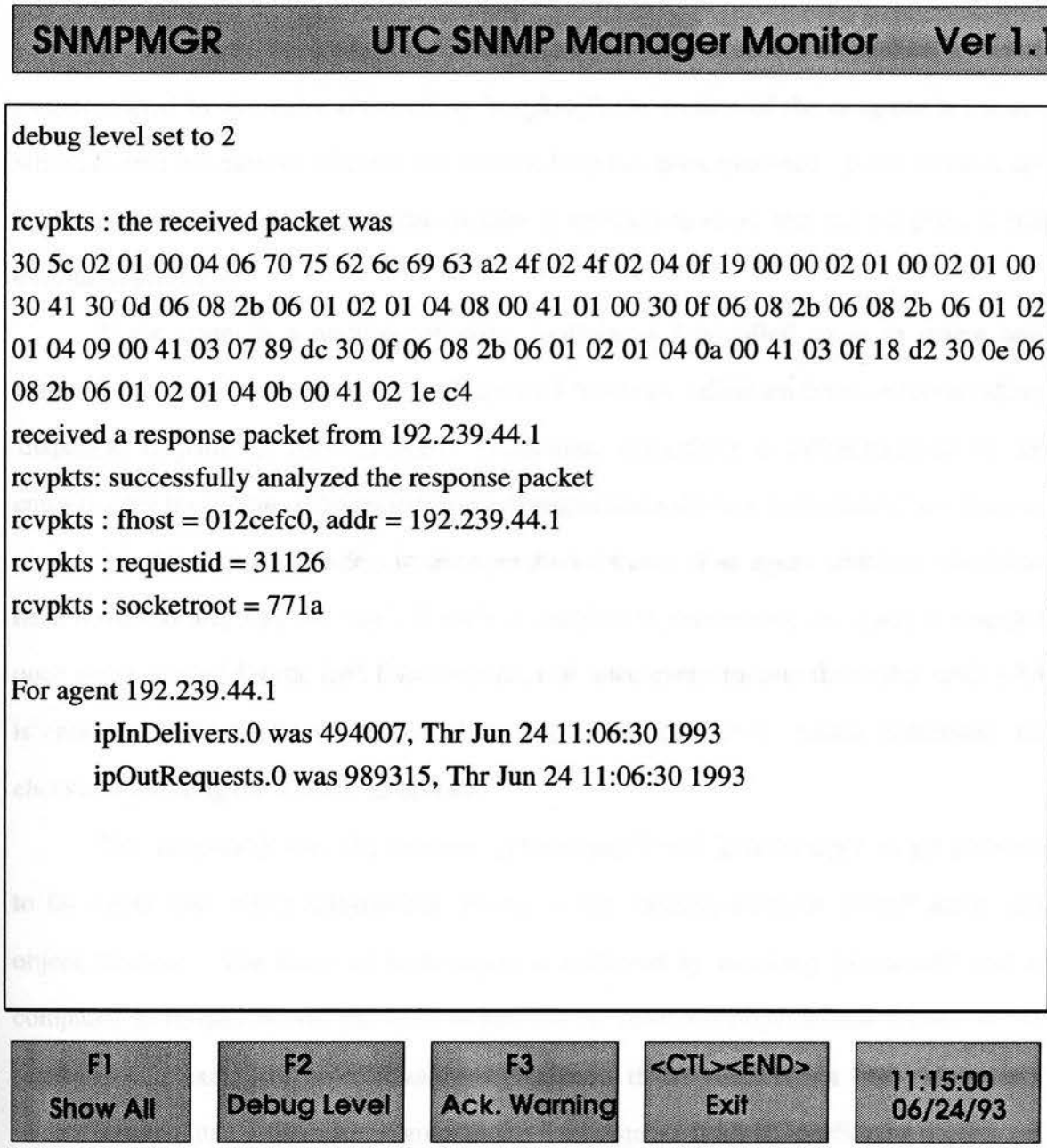


Figure 8. Example Debug Screen.

When function key 3 (**F3**) has been pressed, any alarms that might be sounding (described below) are disabled. If **CTRL-END** (EXIT) has been hit, the variable G_PROGRUN is set to FALSE to notify the program to exit gracefully. If there was not any key stroke, control is given to 'rcvpkts()'.

The 'rcvpkts()' checks to see if any unsolicited packets, such as SNMP trap messages, have been received. If there were no execution errors and no packets received, as determined by the value returned by 'rcvpkts()', the section of the program is entered which counts the number of times the current loop has been executed. If the count is not a multiple of sixty, the clock on the display is updated to show that the program is still executing properly.

If the count is a multiple of sixty, 'pollagents()' is called again to obtain new values for the database objects. If 'pollagents()' has been called ten times without calling 'dispall()', 'dispall()' is also executed. Otherwise, 'dispchgs()' is called to show on the console only those object values that have changed since the last 'pollagents()' invocation.

An alarm situation refers to the operational status of an agent interface which has been retrieved and was not "up". If such a situation is discovered, an alarm is sounded once every second for the first five seconds, and once every minute thereafter until a **F3** is entered on the keyboard, or the alarm condition goes away. Alarm conditions are checked by calling the routine 'ckopstat()'.

The 'ckopstat()' uses the routines 'getnxtroutp()' and 'getnxtvarp()' to get pointers to the agent and object information blocks in the memory-resident SNMP agent and object database. The name of each object is retrieved by invoking 'getvarstr()' and is compared to **ifOperStatus**, the MIB object for an interface's operational status. If the comparison is exact, the object's value is checked. If the value is not "up", 'ckopstat()' returns a minus one; otherwise, it goes to the next object. If all **ifOperStatus** objects are "up", a zero is returned.

If 'ckopstat()' returns a non-zero value, then 'main()' calls 'beep()' to sound the alarm. When this completes, the loop variable G_PROGRUN is checked. If it is TRUE, execution returns to the beginning of the loop. If it is FALSE, the loop is exited and 'endsnmp()' is called. This gracefully terminates independent functions, such as the packet driver, so that the main program can exit without leaving the packet driver in a state which might interfere with the operating system.

Testing

The program was tested on several PCs in Grote Hall room 208. This room was suggested by the UTC network engineer Mike Meyer. IP addresses for the PCs and the agent, along with the community string of the agent, were provided by Mr. Meyer. The program was tested with the objects configuration file shown in Appendix C. Since UTC currently has only one device (router) that supports SNMP, the test was done for that device only. The results of the tests on all four PCs in room 208 were the same for UTC's router .

The object **ifNumber** was tested first to find the number of interfaces on the router and the result was :

ifNumber.0 was 8, Thr Jun 24 11:06:30 1993

The tag '.0' on the object name indicate the object did not have an index. It returned 8 as the number of interfaces for the device with a time stamp to show the date and time of the object value. Therefore, the operational status of all eight interfaces was tested and the following results were obtained for interfaces 1, 2, 3, 5 and 6.

ifOperStatus.1 was Up, Thr Jun 24 11:07:30 1993

ifOperStatus.2 was Up, Thr Jun 24 11:07:30 1993

ifOperStatus.3 was Up, Thr Jun 24 11:07:30 1993

ifOperStatus.5 was Up, Thr Jun 24 11:07:30 1993

ifOperStatus.6 was Up, Thr Jun 24 11:07:30 1993

This result showed five interfaces were up and running. The following results were obtained for interfaces 4, 7 and 8.

ifOperStatus.4 was down, Thr Jun 24 11:08:30 1993

ifOperStatus.7 was down, Thr Jun 24 11:08:30 1993

ifOperStatus.8 was down, Thr Jun 24 11:08:30 1993

These interface cards were not running at the time of testing. Therefore, their operating status was displayed as down. As soon as the operation status of these interfaces was displayed, the alarm system started making the beeping sound to notify that the interfaces were not operating. The alarm was acknowledged by pressing **F3** key, and the beep sound was not made after that.

The object **sysUpTime** was tested for different time durations. Since this object displays the number of seconds the device has been up and running, it is constantly changing in value. The program is designed to display the value of the object once a minute (if the object changes value). The following results were obtained for a period of two minutes.

sysUpTime.0 was 52274315, Thr Jun 24 11:17:30 1993

sysUpTime.0 was 52274375, Thr Jun 24 11:18:30 1993

sysUpTime.0 was 52274435, Thr Jun 24 11:19:30 1993

The value of **sysUpTime** changed by a factor of 60 every time. Since there is 60 seconds in every minute the result was as expected.

The Objects **ifType** and **ifSpeed** were tested to find types and speeds of each interface. The results were as following:

ifType.1 was ethernet-csmacd, Thr Jun 24 11:27:30 1993

ifSpeed.1 was 10000000, Thr Jun 24 11:17:30 1993

ifType.2 was proPointToPointSerial, Thr Jun 24 11:27:30 1993

ifSpeed.2 was 448000, Thr Jun 24 11:17:30 1993

ifType.3 was ethernet-csmacd, Thr Jun 24 11:27:30 1993

ifSpeed.3 was 10000000, Thr Jun 24 11:17:30 1993

ifType.4 was proPointToPointSerial, Thr Jun 24 11:27:30 1993

ifSpeed.4 was 1544000, Thr Jun 24 11:17:30 1993

ifType.5 was ethernet-csmacd, Thr Jun 24 11:27:30 1993

ifSpeed.5 was 10000000, Thr Jun 24 11:17:30 1993

ifType.6 was proPointToPointSerial, Thr Jun 24 11:27:30 1993

ifSpeed.6 was 320000, Thr Jun 24 11:17:30 1993

ifType.7 was ethernet-csmacd, Thr Jun 24 11:27:30 1993

ifSpeed.7 was 10000000, Thr Jun 24 11:17:30 1993

ifType.8 was proPointToPointSerial, Thr Jun 24 11:27:30 1993

ifSpeed.8 was 1544000, Thr Jun 24 11:17:30 1993

It showed interfaces 1, 3, 5 and 7 as Ethernet cards with a speed of 10 Mbits per seconds. Also it displayed interfaces 2, 4, 6 and 8 as serial interfaces with different speed configurations.

A stress test was done to see how many objects the program can handle at the same time. There was not any problem with the number of objects used in the objects configuration file. But, it was very important to maintain the objects configuration file in the correct format. If the objects configuration file did not have the correct format, the program was not be able to create the objects database in memory, and it would exit the program with an error message. The complete objects configuration file that was used for testing the program is included in Appendix C . The results of the tests are shown in Appendix C along with the objects' names as comments (starting with #). The results of the tests were confirmed by Mr. Meyer as being correct, indicating that the program was functioning properly.

Chapter 6

Summary

Network Management with SNMP

As the size of the data network increases, the need for network management tools increases also. SNMP is one of the industry standards for network management. It offers comprehensive services in most areas of network management. SNMP has several advantages compared to other network management protocols.

SNMP assumes only a basic connection less mode access to underlying layers. The OSI Common Management Information Protocol (CMIP), uses a connection oriented model. Both SNMP and CMIP use request-reply interactions. There are fewer operations with SNMP, because functionality of the **action, create,** and **delete** operations of the OSI Common Management Information Service (CMIS), can be performed by the single SNMP **set** operation. The trade-off here is one of defining complexity in the operators or operands. The SNMP approach focuses on a few simple operators with complicated operands, whilst the CMIP approach focuses on more numerous, more complicated operators with simple operands.

Although SNMP is a powerful protocol for network management, it has a few drawbacks. The original RFC 1098 defines the SNMP for IP network only. IP is a widely used datagram protocol for networking. However, not all networks rely on IP for delivery. This is one of the reasons why the Internet community recognizes SNMP as a temporary solution to the need for a standard network management protocol. A possible solution to this problem is the use of SNMP proxy agents. SNMP proxy agents can gather information from network devices that do not speak IP and then convey this information to a management station with SNMP, thus allowing non-IP network devices to be managed via SNMP. A disadvantage of SNMP proxy agents is that specific software must be written for each non IP protocol network device. Another possible

solution is to implement SNMP on top of other transport protocols, thus allowing it to work in networks that do not support IP.

Another problem is that SNMP can be inefficient in retrieving large tables of data. SNMP retrieves tables with Get-Next-Request message. However, using this message to retrieve large tables of data can burden both the network and the destination agent, because the manager station needs to send one request message for each entity in the table in each row. My suggestion to this is having a Get-Column or Get-Row operator to retrieve each column or row of the table with one message instead of multiple messages for each row or column. For example, suppose a table with 2000 rows and 4 columns is to be retrieved. With current operators 2x4x2000 (16000) packets need to be sent on the network to retrieve the table. But if there was a Get-Row operator, only 2x2000 (4000) packets need to be sent on the network to retrieve the table. It reduces the traffic on the network significantly.

The last problem with SNMP is the security of the community string. The SNMP protocol does not provide information or allow configuration changes of a network device without some sort of validation of the ID of the originator of the message. The community string is used as a security password to access the device object. This string is sent within SNMP packets in clear ASCII text. With very little effort (with a device showing the traffic on a network), a person can learn the community string used by a given SNMP agent. This easy access presents a potential security problem in SNMP protocol. My suggestion is to have data encryption for the community string which is recognizable (such as a key for encryption) by the agent and the manager station.

Future of SNMP

The philosophy of SNMP was influenced by the concepts of the Internet-standard Network Management Framework. Both SNMP and management framework are largely derived from SGMP (Simple Gateway Monitoring Protocol [12]). SNMP was

purposefully designed to be a simple protocol, requiring few resources to implement efficiently.

CMIP was developed by the ISO with different goals than the SNMP. SNMP was originally intended for use by IP devices only, while CMIP was intended to be non-protocol specific and for use in the management of all network devices. SNMP was the IAB short-term plan for network management protocol, while CMIP was their long term plan for future requirement. Therefore, the further development of SNMP will be aimed toward keeping the protocol as close as possible to CMIP protocol and eventually toward having CMIP as the only standard in the Internet community.

The Network Management Tool (SNMPMGR)

The Simple Network Management Protocol is becoming widely supported by vendors of Internet Protocol network interconnection devices as a vehicle for managing the operation of those devices. The management tools provided by the vendors, however, are usually designed to manage only the vendor's equipment. Because they are designed to be very visual and elegant, they frequently run only on specific computers, such as expensive workstations. A few vendor independent management packages are available, but these also require specific workstations for execution. For example, the Cisco System provides a SNMP network management tool called CiscoWorks. It requires a SUN workstation with 32MB of RAM, very large disk space (1GB) and is very expensive ($9995). CiscoWorks uses the Cisco specific commands to communicate with the agents. Therefore, it works only with Cisco products.

Novell (NetWare) provides an SNMP network management tool with their file server package. It runs only on the Novell file server. It will gather information concerning the health of the file server only. A five user NetWare costs around $1095.

This project (SNMPMGR) was undertaken to develop a management tool for the University of Tennessee at Chattanooga campus network which runs on a personal

computer. The program accesses network devices, reads values of specific device objects, and prints the values of those objects on the console regardless of the brand of the devices. The devices can be gateways, routers, bridges, host network interfaces -- any device which maintains tables of MIB objects as defined in Internet RFC 1066, "Management Information Base for Network Management of TCP/IP based Internets", and which supports the SNMP. The objects can be the operational state of a specific interface, the input error count of an interface (whether or not a device can forward network packets), etc. The program is vendor independent, because it does not use any vendor specific commands to communicate with the agents. It simply uses the SNMP messages to request information from the agents. Therefore program can be used on any devices that support the SNMP protocol (network management standard protocol). Since UTC has only one device (router) that support SNMP, SNMPMGR was tested on that device only.

The usefulness of this application is that the data it acquires as it runs can be analyzed by someone to determine if a network malfunction has occurred, or if problems may be imminent because error counts have exceeded specific thresholds. With the same SNMP functions that this application uses, other programs can be written so that a network manager will no longer have to wait for a user to report difficulties. Problems can be avoided before they occur rather than corrected after they occur by looking at historical information and predicting maintenance such an interface card is not functioning properly. This application has been implemented only on hosts connected to Ethernets because that type of network is used at the University of Tennessee at Chattanooga.

Suggestion for Future Development

SNMPMGR can be expanded in the following ways.

1) A Graphical User Interface (GUI) can be added to the program to allow runtime changes in the objects being monitored. The GUI can be used to display the health of a device in graphical format such as bar chart or pie chart.

2) The program can be modified to provide an archiving system for objects. The archive files can be used for historical analysis of the health of a device.

3) A report writer can be added to the program that produces a customized reports from the archive file for the network managers.

Glossary

Glossary

abstract syntax : a description of a data type that is independent of machine-oriented structures and restrictions.

ASN.1 : Abstract Syntax Notation One, the OSI language for describing abstract syntax.

access mode: the level of authorization implied by an SNMP community.

Accounting Management : The process of gathering statistics about resources on the network, establishing metrics, checking quotas, determining costs, and billing network clients.

address : The location of a network device or service, for example, an IP address or socket for a program.

agent : A set of software in a network device that is responsible for handling requests by a particular protocol, such as SNMP.

alarm : A sound or message used to grab the attention of a network engineer.

ARP : Address Resolution Protocol, The protocol in the Internet suite of protocols used to map IP addresses onto Ethernet (and other media) addresses.

ANSI : American National Standard Institute. The U.S. national standardization body. ANSI is a member of ISO.

Application Layer : The seventh layer of ISO model that responsible for managing communication between applications.

BER : Basic Encoding Rules, the OSI language for describing transfer syntax.

checksum : An arithmetic sum used to verify data integrity.

CMIP : Common Management Information Protocol, the OSI protocol for network management.

CMIS : Common Management Information Service, the application service element responsible for exchanging network management information.

community : An administrative relationship between SNMP entities.

community string : An ASCII string used for authentication by SNMP.

Configuration Management : The process of obtaining information from network devices and using it to manage their setup.

connection-less mode : A service in which the network delivers data between two systems independent of other simultaneous communication.

connection-oriented mode : A service that has three distinct phase: *establishment,* in which two or more users are bound to a connection; *data transfer,* in which data is exchanged between the users, and, *release,* in which the binding is terminated.

datagram : A piece of data sent to a network that provides connection-less service.

data link layer : The second layer of the OSI Reference Model; responsible for addressing, transmission, error detection, and framing on a channel.

DoD : Department of Defense, A branch of the United States Government responsible for the nation's defense; responsible for developing the DOD protocols, such as TCP/IP.

dotted quad notation : A convention for writing IP address in textual format.

EGP : Exterior Gateway Protocol, a routing protocol for passing reachability information between autonomous systems.

encryption : The scrambling of data through the use of an algorithm.

Ethernet : A networking protocol developed originally by Xerox Corporation for use on LANs.

Fault Management : The process of identifying network faults, isolating the cause of the fault, and if possible, correcting the fault.

fragment : A piece of a packet that has been broken into smaller units.

fragmentation : The process of breaking an IP datagram into smaller parts, such that each fragment can be transmitted in whole on a given physical medium.

gateway : A network device that can perform protocol conversion from one protocol stack to another.

host : A computer system on a data network (the end-system).

IAB : Internet Activities Board, A group that oversees the work in networking technology and protocols for the TCP/IP internetworking community.

interface : A connection between two network devices or hosts.

Internet : A large collection of connected networks running the Internet suite of protocols.

internet : Internet usage, a network in the OSI sense.

internetwork : A collection of networks interconnected by network devices that generally act as a single network.

Internet suit of protocols : a collection of computer-communication protocols originally developed under DoD sponsorship. The Internet suite of protocols is currently the de facto solution for open networking.

IP : Internet Protocol, a network layer protocol that contains addressing and control information for packets to routed.

ISO : International Organization for Standardization, an international body that develops, suggests, and names standards for network protocols.

ISO Reference Model : A network architecture model developed by the ISO; used universally for understanding and teaching network functionality.

IP address : A 32-bit quantity used to represent a point of attachment in an internet.

LAN : Local Area Network, A high-speed network covering a limited geographic area, such as a single building.

managed node : A network device that can be managed by a network management protocol.

MIB : Management Information Base, a database of managed objects accessed by network management protocols.

MIB-I : The first MIB defined for managing TCP/IP-based internets.

MIB-II : The current standard MIB defined for managing TCP/IP-based internets.

network access point : A location, such as a port or software program, by which users obtain access to the network.

Network Layer : The third layer of OSI systems responsible for data transfer across the network, independent of both the media comprising the underlying subnetworks and the topology of those subnetworks.

network management protocol : The protocol used to convey management information.

network protocol : A protocol that operates at the network layer.

octet : An 8-bit or a byte in communication industry.

open system : A system running the OSI protocol stack.

OSI : Open System Interconnection, an international effort to facilitate communications among computers of different manufacture and technology.

packet : A logical collection of data.

PDU : Protocol Data Unit, a data object exchanged by protocol machines, usually containing both protocol control information and user data.

Physical Layer : The first layer of the ISO Reference Model; defines the mechanical, physical, and electrical interface to a network and its associated medium.

polling : The process whereby one device queries other devices for information.

port number : An interface to a network device which identifies an application entity to a transport service in the Internet suite of protocols.

Presentation Layer : The sixth layer of the ISO Reference Model; controls the syntax of information passed between two Application Layer programs.

protocol : A formal description of a set of rules and conventions that describe how network devices exchange information.

proxy agent : An agent that can gather information about other systems and then relay this information to a management station via a protocol, such as SNMP.

reassembly : The process of recombining fragments, at the final destination, into the original IP datagram.

repeater : A Physical Layer network devices that regenerates and propagates bits between two network segments.

RFC : Request For Comments, the document series describing the Internet suite of protocols and related experiments.

router : A network device that can decide how to forward packets through a network by examining Network Layer information.

Security Management : The process of protecting access to sensitive information found on systems attached to a data network.

segment : The unit of exchange in the TCP.

Session Layer : The fifth layer of the ISO Reference Model; coordinates session activities between applications, such as remote procedure calls.

SGMP : Simple Gateway Monitoring Protocol, a network management protocol that was considered for Internet standardization and later evolved into SNMP.

SMI : Structure of Management Information, the rules used to define the objects that can be accessed via a network management protocol (MIB).

SMTP : Simple Mail Transfer Protocol, an Internet protocol providing electronic mail services.

SNMP : Simple Network Management Protocol, A network management protocol used for managing IP network devices.

socket : A software data structure that provide a communication access point within a network device.

subnet mask : A 32-bit quantity indicating which bits in an IP address that identify the physical network.

TCP : Transmission Control Protocol, a Transport Layer protocol that provides reliable transmission of data on IP networks.

TCP/IP : The two most popular Internet protocols that provide Transport Layer and Network Layer services.

Transport Layer : The fourth layer of the ISO Reference Model, responsible for reliable network communication between hosts.

trap : Unsolicited message sent by an SNMP agent to a management station to alert about a specific network event.

UDP : User Datagram Protocol, a connection-less transport protocol used on IP networks.

variable : In SNMP usage means a pairing of an object instance name and associated value.

Workstation : A mid-sized computer system designed for a small number of user.

Bibliography

Bibliography

[1] IAB. "IAB Official Protocol Standards." Request for Comments 1140, DDN Network Information Center, SRI International, May, 1990.

[2] Jeffery D. Case, Mark S. Fedor, Martine L. Schoffstall, and James R. Davin. "A Simple Network Management Protocol." Request for Comments 1157, DDN Network Information Center, SRI International, May, 1990.

[3] Jon B. Postel. "User Datagram Protocol." Request for Comments 768, DDN Network Information Center, SRI International, August, 1980.

[4] Jon B. Postel. "Transmission Control Protocol." Request for Comments 793, DDN Network Information Center, SRI International, September, 1981.

[5] Joyce K. Reynolds, Jon B. Postel. "Official Internet Protocols." Request for Comments 1011, DDN Network Center, SRI International, May 1987.

[6] Keith McCloghrie and Marshall T. Rose. "Management Information Base Network Management of TCP/IP Based Internets." Request for Comments 1156, DDN Network Information Center.

[7] Leinwand Allan and Fang Karen. "Network Management a Practical Perspective" Addison-Wesley Publishing Company, New York, 1993.

[8] Marshall T. Rose. "The Open Book: A Practical Perspective on Open System Interconnection." Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[9] Marshall T. Rose and Keith McCloghrie. "Structure and Identification of Management Information for TCP/IP Based Internets." Request for Comments 1155, DDN Network Information Center.

[10] Marshall T. Rose . "Management Information Base Network Management of TCP/IP Based Internets: MIB-II." Request for Comments 1158, DDN Network Information Center SRI International, May, 1990.

[11] Marshall T. Rose. "The Simple Book: An Introduction to Management of TCP/IP Based Internet." Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[12] Robert T. Barden, Jon B. Postel. "Requirements for Internet gateways." Request for Comments 1009, DDN Network Information Center, SRI International, June, 1987.

[13] Robert V. Crosson "A General Purpose Packet Driver Interface Routines for IBM PC Compatible Computers." Advanced Distributed Computing Systems Group. January 1991.

Appendix A

```
/*----- main() ----------------------------------------------------*/
/* Listen for Simple Network Management Protocol trap packets from
the SNMP agent.  If nothing is heard from it for one minute,
interrogate it to find out if it is still alive.

*/

/*----- include files ---------------------------------------------*/

#include   "common.h"              /* get  structures and definitions */

#define DEFSTORAGE
#include      "\gary\snmp\snmp_src\screen\snmpmgr.h"
#undef DEFSTORAGE

#include      <dos.h>
#include      <conio.h>
#include   <stdio.h>  /* for console i/o */

#include   <malloc.h>              /* for malloc(), free() functions */
#include   <stdlib.h>              /* for malloc(), free(), atoi() functions */

#include   <time.h>                /* for 'localtime()' and 'asctime()' */

/*----- miscellanous definitions ----------------------------------*/
#define    BUFFERSIZ                    64          /* character buffer size */
#define    MIBFILE                      "std.mib"

int univerr = 0;


main( int argc, char *argv[])
{                          /* listen for UDP packets from the agent */
   extern int univerr;
   extern int verbose;
   int wait( unsigned int);
   int initsnmp( char *, char **);
   int rcvpkts( void);
   int endsnmp( void);
   int pollagents( void);
   int dispchgs( void);
   int ckopstat( void);
   char *asctime( const struct tm *);
   struct tm *localtime( const time_t *);
   void free( void *);
   void *malloc( size_t);
   time_t time( time_t *);
   time_t curtime;

   int endflag,                     /* packet receive loop flag */
       status, tocnt, printto,
       i, j, dbase, mibcnt, warnflag;

   char **mibptr, **ptrptr;


            // Create a stack

   _stklen = 65500U;


   if( argc < 2) {
            fprintf( stderr, "%s: enter SNMP database file name\n", PROGNAME);
            fprintf( stderr, "%s: usage: %s snmpdbname \n", PROGNAME, PROGNAME);
            exit( 1);
   }
   verbose = warnflag = dbase = mibcnt = j = 0;
   warnenable = 1;

   mibcnt = 1;
```

```
if( ( mibptr = ( char **) malloc( ( size_t) ( ( sizeof( char *)) * ( mibcnt + 1)))) == NULL)
{
        fprintf( stderr, "%s: memory allocation error\n", PROGNAME);
        exit( 1);
}

ptrptr = mibptr;

*ptrptr = MIBFILE;

 i = initsnmp( argv[1], mibptr);

if( i < 0)
{
        fprintf( stderr, "%s: couldn't initialize the snmp routines\n", PROGNAME);
        endsnmp();

        if( mibptr != NULL)
          free( ( void *) mibptr);

        exit( 1);
}

init_win();

G_pgmrun    = TRUE;

set_maindisp();
get_time_now ();
disp_time ();

cprintf( "interrogating agents\n\r");
if( ( status = pollagents()) < 0)
{
        fprintf( stderr, "%s: can't interrogate the agents, error %d\n" PROGNAME, status);
        endsnmp();
        free( ( void *) mibptr);
        exit( 1);
}

dispall();

printto = tocnt = endflag = 0;


while (G_pgmrun)

{
        // at top of loop, update clock if onscreen

        get_time_now ();

        disp_time ();

 // process any outstanding keystrokes
        key_main();

        /* start receiving packets */

       if( ( status = rcvpkts()) < 0)
       {        /* error on receiving packets? */

           fprintf( stderr, "%s: error %d on receipt of packets\n", PROGNAME, status);
        }
       else
       {
            if( status == 0)
            {                   /* no errors, no packets received? */

                tocnt++;
```

```
                if( tocnt % 60 == 0 )
                {       /* time to interrogate the agents? */

                        if( ( status = pollagents()) < 0)
                        {
                                fprintf( stderr,
                                 "%s: can't interrogate the agents, error %d\n", PROGNAME, status);
                        }

                        tocnt = 0;
                        printto++;

                        if( printto % 10 == 0)
                                dispall();

                        else
                                dispchgs();
                }

            }
        }


        if( ( warnflag = ckopstat()) == 0) /* check operational status */
                warnenable = 1;                 /* enable warning if op status is okay */

        if( ( warnenable > 0) && ( warnflag > 0))
        {
        /* warning enabled and op status not okay? */

                if( warnenable < 6)
                {
                /* yes, sound a waring once a sec.? */

                        beep();     /* yes, sound bell */
                }
                else
                {
                /* sound warning once a minute */
                        if( ( warnenable % 60) == 0)
                                beep();
                }

                warnenable++;           /* increment the bell count */
        }

} // while G_PGMRUN;       /* loop while exit flag is clear */


cleanexit();

endsnmp();
free( ( void *) mibptr);

exit( 0);
}
```

```
/*----- dispchgs() ----------------------------------------*/
/* Show any information in the database that has changed since the last
time the data was shown.

*/

int dispchgs()
{
   int dispinfo( int);
   return( dispinfo( 0));
}

/*----- dispall() ----------------------------------------*/
/* Show all information in the database whether or not it has been
shown before.

*/

int dispall()
{
   int dispinfo( int);
   return( dispinfo( -1));
}

/*----- dispinfo() ----------------------------------------*/
/* If flag is zero, print on the standard output only that data that
has changed since it was last read.  If flag is non-zero, print all the
data.

*/

int dispinfo( flag)
int flag; {
   struct routinfo *getnxtroutp( struct routinfo *);
   struct varinfo *getnxtvarp( struct routinfo *, struct varinfo *);
   int getroutaddr( struct routinfo *, char *, int);
   int getvarstr( struct varinfo *, char *, int);
   int getinxstr( struct varinfo *, char *, int);
   int getvarval( struct varinfo *, void *, int);
   int clrvarchfl( struct varinfo *);
   unsigned long getvartime( struct varinfo *);
   char *getvartype( struct varinfo *);
   char *xlatevalue( struct varinfo *);
   char *asctime( const struct tm *);
   struct tm *localtime( const time_t *);
   struct routinfo *rinfop;
   struct varinfo *vblkp;
   int headerflag, i;
   unsigned long li;
   char tempbuf[ BUFFERSIZ], *strp;

   if( ( rinfop = getnxtroutp( NULL)) == NULL)
   {
            cprintf( "There is no agent information\n\r");
   }

   while( rinfop != NULL)
   {
            headerflag = 0;
            vblkp = NULL;
            while( ( vblkp = getnxtvarp( rinfop, vblkp)) != NULL)
            {
                     if( ( flag != 0) || ( getvarchfl( vblkp) != 0))
                     {
                              if( headerflag == 0)
                              {
                                       headerflag++;

                                       if( getroutaddr( rinfop, tempbuf, BUFFERSIZ) < 0)
                                                cprintf( "dispinfo: can't get agent string\n\r");
```

```
            cprintf( "\r\n\nFor agent %s\n\r", tempbuf);
}
if( ( i = getvarstr( vblkp, tempbuf, BUFFERSIZ)) < 0)
            cprintf( "   Can't get variable, error %d\n\r", i);

cprintf( "   %s", tempbuf);

if( ( i = getinxstr( vblkp, tempbuf, BUFFERSIZ)) < 0)
            cprintf( "\n\r   Can't get index, error %d\n\r", i);

if( i == 0)   /* was there an index? */
            cprintf( " was ");          /* no, don't print an index */
else                        /* there was an index, print it */
            cprintf( ".%s was ", tempbuf);

if( ( strp = xlatevalue( vblkp)) != NULL)
{
            cprintf( "%s, ", strp);
 }
else
{
            if( ( strp = getvartype( vblkp)) == NULL)
                        cprintf( "\n\r   Can't get variable type\n\r");
            else
            {
                        for( i = 0 ; strp[ i] != '\0' ; i++);
                                    if( cmpstr( strp, "unsigned long", i) == 0)
                                    {
                                                if( getvarval( vblkp, ( void *) &li, sizeof( li))< 0)
                                                   fprintf( stderr, "\n   Can't get long integer value\n");
                                                else
                                                {
                                                            cprintf( "%lu, ", li);
                                                }
                                    }
                                    else
                                    {
                                                if( cmpstr( strp, "int", i) == 0)
                                                {
                                                        if( getvarval( vblkp, ( void *) &i, sizeof( i)) < 0)
                                                            fprintf( stderr, "\n   Can't get integer\n");
                                                        else
                                                            cprintf( "%d, ", i);
                                                }
                                                else
                                                {
                                                  if( cmpstr( strp, "hex string", i) == 0)
                                                if( getvarval( vblkp, ( void *) tempbuf, BUFFERSIZ) < 0)
                                                   fprintf( stderr,"\n   Can't get hex string\n");
                                                else
                                                {
                                                  cprintf( "%s, ", tempbuf);
                                                 }
                                    }
                                    else
                                    {
                                    if( cmpstr( strp, "ASCII string", i)== 0)
                                    {
                                       if( getvarval( vblkp, ( void *) tempbuf,BUFFERSIZ) < 0)
                                                fprintf( stderr,"\n   Can't get ASCII string\n");
                                       else
                                       {
                                          cprintf( "%s, ", tempbuf);
                                       }
                        }
                        else
                        {
                                    if( cmpstr( strp, "integer string", i) == 0)
                                       {
                                    if( getvarval( vblkp,( void *) tempbuf,          BUFFERSIZ) < 0)
                                                fprintf( stderr,"\n   Can't get integer string\n");
```

```
                                                                else
                                                                    {
                                                                        cprintf( "%s, ",tempbuf);
                                                                    }
                                                                }
                                                                else
                                                                    fprintf( stderr,"\n   Unsupported variable type \"%s\"\n",strp);
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                        /* now add the time stamp to the line */

                                        if( ( li = getvartime( vblkp)) < 0L)

                                            fprintf( stderr, "\n   Can't get time stamp\n");
                                        else
                                        {
                                            if( li == 0L)
                                                cprintf( "timestamp is 0\n\r");
                                            else
                                                cprintf( "%s\r",asctime( localtime( ( time_t *)&li)));
                                        }

                                        if( clrvarchfl( vblkp) < 0)                 /* cleared this variable's flag? */
                                            fprintf( stderr, "\tCan't clear the change flag\n");
                                    }
                                }
                                rinfop = getnxtroutp( rinfop);
                            }
                            return( 0);
}

/*----- ckopstat() ----------------------------------------------------*/
/* Check all variables in the database for ifOperStatus. If one is
found and its status is not 'up', return one. Otherwise return zero
unless there is an error; then return ERROR.

*/
int ckopstat( void) {
    struct routinfo *getnxtroutp( struct routinfo *);
    struct varinfo *getnxtvarp( struct routinfo *, struct varinfo *);
    int getvarstr( struct varinfo *, char *, int);
    int cmpstr( unsigned char *, unsigned char *, int);
    char varname[ STRINGSIZ];
    struct routinfo *rinfop;
    struct varinfo *vinfop;
    rinfop = getnxtroutp( NULL);   /* get first agent pointer */

    while( rinfop != NULL)
    {
            vinfop = getnxtvarp( rinfop, NULL);          /* get first variable ptr */

            while( vinfop != NULL)
            {
                    if( getvarstr( vinfop, varname, STRINGSIZ) < 0)
                    {
                            fprintf( stderr, "ckopstat: can't get variable name\n");
                            return( ERROR);
                    }

                    if( cmpstr( ( unsigned char *) varname, ( unsigned char *) "ifOperStatus", 13) == 0)
                    {
                            if( *( ( int *) vinfop->varvalue) != 1)
                                    return( 1);
                    }

                    vinfop = getnxtvarp( rinfop, vinfop); /* get next variable ptr */
            }
            rinfop = getnxtroutp( rinfop);      /* get next agent pointer */
```

```
}
  return( 0);
}
// END SNMPMGR.C
```

```
/*----- getcmunp() ----------------------------------------------------*/
/* Given a network descriptor, return the pointer to the SNMP community
associated with that nd.  Return NULL on error.

*/

#include    "/gary/snmp/snmp_src/pd/pdnet.h"
#include    "/gary/snmp/snmp_src/pd/sockmgr/sockmgr.h"
#include    <stdio.h>                   /* for NULL */

unsigned char *getcmunp( int nd)
{
            extern struct sockdata *socketroot;
   struct sockdata *sockdatap;

            sockdatap = socketroot;

            while( ( sockdatap->nd != nd) && (sockdatap->nxtstrcp != NULL))
            sockdatap = sockdatap->nxtstrcp;

            if( sockdatap->nd != nd)
            {
                        fprintf( stderr, "getcmunp: no sockdata for nd %d\n", nd);
                        return( NULL);
            }

            return( ( unsigned char *)&( sockdatap->routinfop->community[0]));
}

//end getcmunp.c
```

```
/*---- getconninfo() ----------------------------------------*/
/* Given a network descriptor, fill in the given array with the
information about the connection associated with that descriptor.
Return error if the network descriptor is not allocated; return 0 on
success.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                /* for NULL, fprintf(), etc. */

int getconninfo( int nd, struct sockdata *sockinfop)
{
  extern struct sockdata *socketroot;
  struct sockdata *sockdatap;

        if( socketroot == NULL)
        {
                fprintf( stderr, "getconninfo: socketroot is NULL\n");
                return( ERROR);
        }

        sockdatap = socketroot;

        while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
                sockdatap = sockdatap->nxtstrcp;

        if( sockdatap->nd != nd)
        {
                fprintf( stderr, "getconninfo: no matching nd found for %d\n", nd);
                return( ERROR);
        }

        sockinfop->nd = sockdatap->nd;
        sockinfop->reqid = sockdatap->reqid;
        sockinfop->routinfop = sockdatap->routinfop;
        sockinfop->fhost = sockdatap->fhost;
        sockinfop->fsocket = sockdatap->fsocket;
        sockinfop->lsocket = sockdatap->lsocket;
        sockinfop->protocol = sockdatap->protocol;

        return( 0);
}
// end getconni.c
```

```
/*----- getnxtsendnd() ----------------------------------------*/
/* Given a send network descriptor, find the next send network descriptor
in the socket information table. If the given network descriptor is -1,
find the first send network descriptor in the table. Return ERROR on
error.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>  /* for NULL, fprintf(), etc. */

int getnxtsendnd( int nd)
{
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap;

        if( socketroot == NULL)
        {                       /* no socket info? */
                return( ERROR);                         /* right, error */
        }

        sockdatap = socketroot;                 /* start from the first info blk */

        if( nd != ( -1))
        {       /* looking for send block following the nd one */
                /* loop while we haven't found the nd block */
                while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
                {
                        sockdatap = sockdatap->nxtstrcp;
                }

                if( sockdatap->nd != nd)
                {       /* found the right block? */

                        fprintf( stderr, "getnxtsendnd: can't find matching nd for %d\n",nd);
                        return( ERROR - 3);             /* no, error */
                }
                if( sockdatap->reqid < ( long)( -1))
                {       /* does this blk have the right request id along with the right nd? */

                        fprintf( stderr, "getnxtsendnd: wrong request id, reqid is %ld\n", sockdatap->reqid);
                        return( ERROR - 4);             /* no, error */
                }

                if( sockdatap->nxtstrcp == NULL)
                {
                        return( ERROR - 6);
                }
                else
                        sockdatap = sockdatap->nxtstrcp;
        }
                /* we found the right send blk, look for the next send block */

        while( ( sockdatap->reqid < ( long)( -1)) && ( sockdatap->nxtstrcp != NULL))
        {
                sockdatap = sockdatap->nxtstrcp;
        }

        if( sockdatap->reqid < ( long)( -1))
        {       /* found a send blk? */
                return( ERROR - 2);                     /* no, error */
        }

        return( sockdatap->nd);                 /* return the nd of the right blk */
}
// end getnxtsn.c
```

```
/*----- getnxttrapnd() ----------------------------------------*/
/* Given a trap network descriptor, return the next trap network
descriptor in the socket information table.  If the network
descriptor passed to this routine is -1, start with the first trap
network descriptor.  Return ERROR on error.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>  /* for NULL, fprintf(), etc. */

int getnxttrapnd( int nd)
{
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap;

        if( socketroot == NULL)
        {                       /* no socket info? */

                return( ERROR);                         /* right, error */
        }

        sockdatap = socketroot;

        if( nd != ( -1))
        {       /* looking for trap block following the nd one */
                /* loop while we haven't found the nd block */

                while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
                {

                        sockdatap = sockdatap->nxtstrcp;
                }

                if( sockdatap->nd != nd)
                {       /* found the right block? */

                        fprintf( stderr, "getnxttrapnd: no matching nd found for %d\n", nd);
                        return( ERROR - 3);             /* no, error */
                }

                if( sockdatap->reqid != ( long)( -2))
                {       /* does this blk have the right request id along with the right nd? */

                        fprintf( stderr, "getnxttrapnd: incorrect request id found, %ld\n", sockdatap->reqid);
                        return( ERROR - 4);             /* no, error */
                }

                if( sockdatap->nxtstrcp == NULL)
                {       /* is there a next block? */

                        return( ERROR - 6);                     /* no, error */
                }
                else                                    /* there is a next block */
                        sockdatap = sockdatap->nxtstrcp; /* point to it */

        }                       /* we found the right trap blk, look for the next trap block */

        while( ( sockdatap->reqid != ( long)( -2)) && ( sockdatap->nxtstrcp != NULL))
        {
                /* loop while we find non-trap blocks */
                sockdatap = sockdatap->nxtstrcp;
        }

        if( sockdatap->reqid != ( long)( -2))
        {       /* found a trap blk? */
                return( ERROR - 2);                     /* no, error */
        }
        return( sockdatap->nd);                 /* return the nd of the right blk */
}
//end getnxttn.c
```

```
/*---- getconnreqid() ----------------------------------------*/
/* Given a network descriptor (nd), return the request id in the
connection information structure.  Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>  /* for NULL, fprintf(), etc. */

long getconnreqid( int nd)
{
        extern int verbose;
        extern struct sockdata *socketroot;
        extern int univerr;
        struct sockdata *sockdatap;

        if( socketroot == NULL)
        {
                fprintf( stderr, "getconnreqid: socketroot is NULL\n");
                univerr = -1;
                return( NULL);
        }

        sockdatap = socketroot;

        while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
        {
                if( verbose == 513)
                        fprintf( stderr, "getconnreqid: found nd = %d, request id = %ld\n",
                                                sockdatap->nd, sockdatap->reqid);
                sockdatap = sockdatap->nxtstrcp;
        }

        if( sockdatap->nd != nd)
        {
                fprintf( stderr, "getconnreqid: no matching nd found for %d\n", nd);
                univerr = -2;
                return( NULL);
        }

        if( verbose == 513)
                fprintf( stderr, "getconnreqid: returning nd = %d, request id = %ld\n", sockdatap->nd, sockdatap->reqid);

        return( sockdatap->reqid);
}
// END GETREQID.C
```

```
/* net.c - network functions
*/

/*---- include files ----------------------------------------------*/
#include   "\gary\snmp\snmp_src\pd\sockmgr\sockmgr.h"
#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>              /* for NULL, fprintf(), etc. */
#include   <dos.h>                /* for FP_OFF(), etc. */
#include   <malloc.h>             /* for malloc(), etc. */

/*---- set_option()-----------------------------------------------*/
/* Enable the option specified for the network descriptor given.

*/

int set_option( int nd, int proto, int option, char far *optionaddr, int optionlen)
{
        return( 0);
}

/*---- net_listen() ----------------------------------------------*/
/* Enable the packet driver for receiving IP packets.

*/

int net_listen( int nd, int proto, struct addr *sockinfop)
{
        extern struct dvrinfo pdvrinfo;
        extern int iphandle;
        extern int neterrno;
        extern void far buffmgr( void);

        int initdvr( struct dvrinfo *, unsigned char *, int, char far *);

        if( iphandle < 0)
        {
                int iptype;
                iptype = BYTESWAP( E_TYPE_IP);
                if( ( iphandle = initdvr( &pdvrinfo, ( unsigned char *) &iptype,
                                        sizeof( iptype), ( char far *) buffmgr)) < 0)
                {
                        neterrno = iphandle;
                        return( ERROR);
                }
        }

        sockinfop->protocol = ( unsigned char) proto;

        return( 0);
}
```

```
/*----- net_readfrom() ----------------------------------------------*/
/* Given a network descriptor, a pointer to a buffer, the length of
that buffer, a pointer to socket data, and a flags integer, check the
appropriate packet queue (as identified by the socket protocol) for
packets meant for the specified network descriptor. If any are found,
copy the data from the packets into the buffer, providing the buffer
is large enough.

*/

int net_readfrom( int nd, char *buffer, unsigned int buflen,
                                struct addr *sockinfop, unsigned int flags)
{
        extern struct bufferblk far *udpqptr, far *tcpqptr, far *icmpqptr;
        extern int neterrno;

        int getpkts( void);
        int fmovestr( unsigned char far *, unsigned char far *, int);

        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct updhdr far *udphdrp;
        struct tcphdr far *tcphdrp;
        unsigned char far *ucp;
        struct bufferblk far *qptr, far *prevbbp;
        int datalen;

        getpkts();

        if( sockinfop->protocol == ( unsigned char) IP_PROTO_UDP)
                prevbbp = qptr = udpqptr;
        else
        {
                if( sockinfop->protocol == ( unsigned char) IP_PROTO_TCP)
                prevbbp = qptr = tcpqptr;
                else
                {
                        fprintf( stderr, "net_readfrom: unrecognized protocol %02x\n", sockinfop->protocol);
                        neterrno = 0;

                        return( ERROR);
                }
        }

        if( FP_OFF( qptr) == NULL)
                return( 0);

        while( ( qptr->nxtblockp != NULL) && ( qptr->handle != nd))
        {
                prevbbp = qptr;
                FP_OFF( qptr) = ( unsigned int) qptr->nxtblockp;
        }

        if( qptr->handle != nd)
                return( 0);

        if( qptr->bufferlen > buflen)
                return( ERROR);

        if( qptr == udpqptr)
                FP_OFF( udpqptr) = ( unsigned int) qptr->nxtblockp;
        else
        {
                if( qptr == tcpqptr)
                        FP_OFF( tcpqptr) = ( unsigned int) qptr->nxtblockp;
                else
                {
                        prevbbp->nxtblockp = qptr->nxtblockp;
                }
        }
```

```
        ethhdrp = ( struct ethhdr far *) qptr->data;

        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);

        datalen = BYTESWAP( iphdrp->ippktlen);   /* get ip pkt len */

        datalen -= F_IP_IHL( iphdrp->ipver_hl);      /* subtract the ip header */

        ucp = ( unsigned char far *) ( ( unsigned char far *) iphdrp +
                                                F_IP_IHL( iphdrp->ipver_hl));

        if( sockinfop->protocol == IP_PROTO_UDP)
        {
                ucp += sizeof( struct udphdr);
                datalen -= sizeof( struct udphdr);
        }
        else
        {
                if( sockinfop->protocol == IP_PROTO_TCP)
                {
                        tcphdrp = ( struct tcphdr far *) ucp;
                        ucp += F_TCP_HDL( tcphdrp->offset);
                        datalen -= F_TCP_HDL( tcphdrp->offset);
                }
        }

        fmovestr( ucp, ( unsigned char far *) buffer, datalen);
        qptr->usedflag = 0;

        return( datalen);
}
```

```
/*----- net_connect() ----------------------------------------------*/
/* Given a network descriptor, the protocol to use in sending packets,
and a pointer to a struct containing the foreign host, foreign socket,
and local socket to use, establish a connection with the foreign
machine.  Only UDP is currently supported.

*/

int net_connect( int nd, int proto, struct addr *sockinfop)
{
        extern int iphandle;
        extern struct dvrinfo pdvrinfo;
        extern int neterrno;

        void far buffmgr( void);
        int net_getdesc( void);
        int initdvr( struct dvrinfo *, unsigned char *, int, char far *);

        if( proto != IP_PROTO_UDP)
                return( ERROR);

        sockinfop->protocol = ( unsigned char) proto;

        if( iphandle < 0)
        {
                int iptype;
                iptype = BYTESWAP( E_TYPE_IP);

                if( ( iphandle = initdvr( &pdvrinfo, ( unsigned char *) &iptype,
                                        sizeof( iptype), ( char far *) buffmgr)) < 0)
                {
                        neterrno = iphandle;
                        return( ERROR);
                }
        }

        if( nd < 0)
        {
                if( ( nd = net_getdesc()) < 0)
                        return( ERROR);
        }

        return( nd);
}
```

```
/*---- net_writeto() ----------------------------------------------*/
/* Given a network descriptor, a pointer to a buffer containing data,
the length of the data buffer, a pointer to socket information, and
any flags, send a packet containing the data in the buffer to the host
specified by the network descriptor and the socket information. Only
UDP protocol is supported currently.

*/

int net_writeto( int nd, char *buffer, unsigned int buflen,
                                        struct addr *sockinfop, unsigned int flags)
{

        extern struct dvrinfo pdvrinfo;

        void *malloc( size_t);
        void free( void *);
        int movestr( unsigned char *, unsigned char *, int);
        unsigned char *getethaddr( unsigned char *);

        struct ethhdr *ethhdrp;
        struct iphdr *iphdrp;
        struct udphdr *udphdrp;
        unsigned char *ucp;
        int i;

        if( sockinfop->protocol != IP_PROTO_UDP)
        {

                fprintf( stderr, "net_writeto: socket protocol not UDP\n");
                return( ERROR);
        }

        if( ( ethhdrp = ( struct ethhdr *) malloc( E_MX_PSIZ)) == NULL)
        {
                fprintf( stderr, "net_writeto: can't get memory for a packet\n");
                return( ERROR);
        }

        if( ( ucp = getethaddr( ( unsigned char *) &( sockinfop->fhost))) == NULL)
        {
                fprintf( stderr, "net_writeto: can't get host ethernet address\n");
                return( ERROR);
        }

        movestr( ucp, ethhdrp->destaddr, E_ADD_SIZ);

        movestr( pdvrinfo.locethaddr, ethhdrp->srcaddr, E_ADD_SIZ);

        ethhdrp->type = BYTESWAP( E_TYPE_IP);

        iphdrp = ( struct iphdr *) ( ethhdrp + 1);

        initiphdr( iphdrp);

        movestr( ( unsigned char *) &( sockinfop->fhost), iphdrp->destaddr, IP_ADD_SIZ);
        udphdrp = ( struct udphdr *) ( ( unsigned char *) iphdrp + IP_IHL (iphdrp->ipver_hl));

        udphdrp->srcport = BYTESWAP( sockinfop->lsocket);

        udphdrp->destport = BYTESWAP( sockinfop->fsocket);

        initudppkt( iphdrp, sizeof( struct ethhdr) + IP_IHL( iphdrp->ipver_hl) + sizeof( struct udphdr) + buflen, buffer, buflen);

        i = sendpkt( ( unsigned char *) ethhdrp, sizeof( struct ethhdr) + BYTESWAP( iphdrp >ippktlen));

        free( ( void *) ethhdrp);
        if( i < 0)
        {
                fprintf( stderr, "net_writeto: error %d on sending packet\n", i);
                return( ERROR);
```

```
        }
        return( 0);
}
```

```
/*----- net_getdesc() ----------------------------------------*/
/*  Return the next available network descriptor.

*/

struct netdesc
{
        struct netdesc *nxtdescp;
        int nd;
} *firstndp = NULL;

int net_getdesc( void)
{
        extern struct netdesc *firstndp;
        void *malloc( size_t);
        struct netdesc *currndp, *nextndp;

        if( firstndp == NULL)
        {
                if( ( currndp = firstndp =
                        ( struct netdesc *) malloc( sizeof( struct netdesc))) == NULL)
                                return( ERROR);

                currndp->nxtdescp = NULL;
                currndp->nd = MIN_ND;
        }
        else
        {
                currndp = firstndp;
                while( (currndp->nxtdescp != NULL) && ( currndp->nxtdescp->nd ==

                currndp->nd + 1))
                {

                currndp = currndp->nxtdescp;
                }

                if( currndp->nxtdescp == NULL)
                        nextndp = NULL;
                else
                        nextndp = currndp->nxtdescp;

                if( ( currndp->nxtdescp =
                        ( struct netdesc *) malloc( sizeof( struct netdesc))) == NULL)
                {
                        if( nextndp != NULL)
                        currndp->nxtdescp = nextndp;

                        fprintf( stderr, "net_getdesc: memory allocation error\n");

                        return( ERROR);
                }

                currndp->nxtdescp->nd = currndp->nd + 1;
                currndp = currndp->nxtdescp;

                if( nextndp != NULL)
                        currndp->nxtdescp = nextndp;
                else
                        currndp->nxtdescp = NULL;
        }

        return( currndp->nd);
}
```

```
/*----- net_releaseall() ----------------------------------------------*/
/* Release all network descriptors currently in use.

*/

int net_releaseall( void)
{
        extern struct netdesc *firstndp;
        extern int arphandle, iphandle;

        void free( void *);
        int relhandle( int);

        struct netdesc *currndp;

        while( firstndp != NULL)
        {
                currndp = firstndp;
                firstndp = currndp->nxtdescp;
                free( ( void *) currndp);
        }

        if( arphandle >= 0)
        {
                relhandle( arphandle);
                arphandle = ( -1);
        }

        if( iphandle >= 0)
        {
                relhandle( iphandle);
                iphandle = ( -1);
        }

        return( 0);
}

/*----- net_release() -------------------------------------------------*/
/* Release the specified network descriptor.
*/

int net_release( int nd)
{
        extern struct netdesc *firstndp;
        void free( void *);
        struct netdesc *currndp, *prevndp;

        if( firstndp == NULL)
        {
                return( 0);
        }

        currndp = firstndp;
        prevndp = NULL;

        while( (currndp->nd != nd) && ( currndp->nxtdescp != NULL))
        {
                prevndp = currndp;
                currndp = currndp->nxtdescp;
        }

        if( currndp->nd != nd)
        {
                return( 0);
        }

        if( prevndp == NULL)
                firstndp = currndp->nxtdescp;
        else
                prevndp->nxtdescp = currndp->nxtdescp;
```

```
        free( ( void *) currndp);

        return( 0);
}

/*---- pneterror() ----------------------------------------*/
/* Print an error message based on the errors in neterrno and
netsuberrno.

*/

void pneterror( char *msg)
{
        extern int neterrno;
        extern int netsuberrno;
        fprintf( stderr, "%s", msg);
}

// END NET.C
```

```
/*---- setconnreqid() ----------------------------------------*/
/* Given a network descriptor (nd), set the request id in the
connection information structure.  Return ERROR on error; zero on
success.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                /* for NULL */

int setconnreqid( int nd, long requestid)
{
        extern int verbose;
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap;

        if( socketroot == NULL)
        {
                fprintf( stderr, "setconnreqid: socketroot is NULL\n");
                return( ERROR);
        }

        sockdatap = socketroot;

        while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
        {
                if( verbose == 514)
                        fprintf( stderr, "setconnreqid: found nd = %d, request id = %ld\n",
                                                                        sockdatap->nd, sockdatap->reqid);
                sockdatap = sockdatap->nxtstrcp;
        }

        if( sockdatap->nd != nd)
        {
                fprintf( stderr, "setconnreqid: no matching nd found for %d\n", nd);
                return( ERROR);
        }

        sockdatap->reqid = requestid;

        if( verbose == 514)
                fprintf( stderr, "setconnreqid: set nd = %d, request id = %ld\n",  sockdatap->nd, sockdatap->reqid);
        return( 0);
}

// END SETREQID.C
```

```
/*----- setrinfop() ----------------------------------------------*/
/* Given a network descriptor, set the pointer to the SNMP community
   associated with that nd. Return ERROR on error; zero on success.
*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                /* for NULL, fprintf(), etc. */

int setrinfop( int nd, struct routinfo *routinfop)
{
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap;
        sockdatap = socketroot;

        while( ( sockdatap->nd != nd) && (sockdatap->nxtstrcp != NULL))
                sockdatap = sockdatap->nxtstrcp;

        if( sockdatap->nd != nd)
        {
                fprintf( stderr, "setrinfop: no sockdata for nd %d\n", nd);
                return( ERROR);
        }

        sockdatap->routinfop = routinfop;

        return( 0);
}

// END SETRINFP.C


/*----- sockglob.c -----------------------------------------------*/
/*----- global variables -----------------------------------------*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"

int univerr,
        netermo,
        netsubermo;

struct dvrinfo pdvrinfo;

int pdvector = 0;


// END SOCKGLOB.C
```

```
/*----- enchainbuf() ------------------------------------------------*/
/*  Add the buffer identified by the handle 'handle', the buffer pointer
    'buffer', and the buffer length 'bufferlen' to the chain of buffers.  A
    buffer block of type 'bufferblk' is created for each buffer.  The buffer
    blocks are chained together with the external pointer 'bufroot' pointing
    to the first of the chained blocks.  This routine is useful only with
    the dummy packet driver and simulates what a packet driver would do with
    a full buffer.
*/

#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                /* for int86x() */

#define    ERROR ( -1)

int enchainbuf( int handle, int bufferlen, unsigned char far *buffer)
{
        extern int pdvector;
        extern int far asmerr;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        struct SREGS segregs;
        union REGS regs;

        fprintf( stderr, "enchainbuf:\n");

        segread( &segregs);                         /* get the segment registers */

        regs.x.ax = 1;                              /* set the function number */
        regs.x.bx = ( unsigned int) handle;         /* set the handle */
        regs.x.cx = ( unsigned int) bufferlen;      /* set the buffer length */
        regs.x.si = FP_OFF( buffer);      /* set the buffer segment addr */
        segregs.ds = FP_SEG( buffer);     /* set the buffer offset addr */

        int86x( pdvector, &regs, &regs, &segregs);  /* call the driver */
        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "enchainbuf: error on int86x call\n");
                return( ERROR);                                     /* yes, error */
        }

        if( asmerr != 0)
        {                       /* any assembly language routine errors? */

                fprintf( stderr, "enchainbuf: asmerr was %d\n", asmerr);
                return( ERROR - 1);             /* yes, return error */
        }

        return( 0);
}

// END ENCHNBUF.C
```

```
/*----- getbuf() ------------------------------------------------*/
/* Get a buffer from the buffer manager routine 'buffmgr' in the same
manner a packet driver would request one. This routine is useful only
with the dummy packet driver dummypd.

*/

#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                        /* for int86x() */

unsigned char far *getbuf( int handle, unsigned int len)
{
        extern int pdvector;
        extern int far asmerr;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        struct SREGS segregs;
        union REGS regs;
        unsigned char far *buffer;

        fprintf( stderr, "getbuf:\n");

        segread( &segregs);             /* get segment registers */
        regs.x.ax = 0;                          /* set the function number */
        regs.x.bx = ( unsigned int) handle;     /* set the handle */
        regs.x.cx = len;                /* set the required buffer length */
        regs.x.di = 0;                          /* clear the buffer offset */
        segregs.es = 0;                         /* clear the buffer segment */
        int86x( pdvector, &regs, &regs, &segregs);  /* call the driver */

        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "getbuf: error on int86x call\n");
                return( ( unsigned char far *) NULL);       /* yes, error */
        }

        FP_SEG( buffer) = segregs.es;           /* get buffer segment */
        FP_OFF( buffer) = regs.x.di;            /* get buffer offset */

        if( buffer == ( unsigned char far *) NULL)
        {       /* no buffer? */

                fprintf( stderr, "getbuf: 0:0 returned, no buffer allocated\n");
                fprintf( stderr, "\tasmerr is %d\n", asmerr);

                return( ( unsigned char far *) NULL);       /* right, error */
        }

        fprintf( stderr, "getbuf: returning buffer at %04x:%04x\n", FP_SEG( buffer),

        FP_OFF( buffer));

        return( buffer);                /* return a pointer to the buffer */
}

// END GETBUF.C
```

```
/*----- getlocaddr() ----------------------------------------*/
/*  Get the local network address associated with 'handle' and put it
in 'buffer', which is 'bufflen' bytes long.

*/

#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                        /* for int86x() */

#define    ERROR                  ( -1)

int getlocaddr( int handle, unsigned char *buffer, int bufflen)
{
           extern int pdvector;
           void segread( struct SREGS *);
           int int86x( int, union REGS *, union REGS *, struct SREGS *);
           char *prdvrerr( int);
           struct SREGS segregs;
           union REGS regs;

           segread( &segregs);                          /* get the segment registers */
           regs.h.ah = 6;                               /* set get_address() function number */
           regs.x.bx = ( unsigned int) handle;          /* set the handle */
           regs.x.cx = ( unsigned int) bufflen;/* set the buffer length */
           regs.x.di = ( unsigned int) buffer;          /* set the buffer addr */
           segregs.es = segregs.ds;                     /* set the buffer segment addr */
           int86x( pdvector, &regs, &regs, &segregs);   /* call the software int. */

           if( regs.x.cflag != 0)
           {                    /* error on software interrupt? */

                      fprintf( stderr, "getlocaddr: error on int86x call\n");
                      fprintf( stderr, "\tdriver error was %s\n", prdvrerr( ( int) regs.h.dh));

                      return( ERROR);                                   /* yes, error */
           }

           return( 0);
}

// END GETLOCAD.C
```

```
/*----- getpdint() ----------------------------------------------*/
/*  Find the software interrupt between 0x60 and 0x80, inclusive,
used by the packet driver.

*/

#include    <stdio.h>           /* for fprintf(), etc. */
#include    <dos.h>                     /* for int86x() */

#define     ERROR               ( -1)

int getpdint( void)
{
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        void ( interrupt far * _dos_getvect( unsigned int))();
        unsigned char far *vecp;
        unsigned int vec =0x60;         /* it was 0x60;*/
        int status;

        while( vec < ( ( unsigned int) 0x81 /*it was 81*/))
        {       /* while the vector we are looking at is within the range, loop */

                vecp = ( unsigned char far *)_dos_getvect( vec);        /* get the vector associated with the vector number */

                if( vecp != NULL)
                {                       /* is the pointer null? */
                                        /* the vector isn't null */
                        if( ( status = fcmpstr( ( unsigned char far *) &( vecp[3]),
                                        ( unsigned char far *) "PKT DRVR", 9))
                                                == 0)
                        {       /* is this the packet driver? */
                                        return( ( int) vec);    /* yes, return the vector number */
                        }
                }

                vec++;                          /* not the packet driver, go to the next one */
        }

        return( ERROR);                 /* no packet driver found, return error */
}

// END GETPDINT.C
```

```
/*----- getdvrinfo() -----------------------------------------------*/
/*  Get information about the packet driver using the software interrupt
'pdvector'.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet driver header */
#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                          /* for int86x() */

int getdvrinfo( struct dvrinfo *dvrinfop)
{
        extern int pdvector;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        char *prdvrerr( int);
        struct SREGS segregs;
        union REGS regs;

        segread( &segregs);                          /* get the segment registers */
        regs.x.ax = 0x1ff;                           /* set the driver_info() function no. */
        regs.x.bx = ( unsigned int) dvrinfop->handle; /* set the handle */
        int86x( pdvector, &regs, &regs, &segregs);   /* call the software int. */
        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "getdvrinfo: error on int86x call\n");
                fprintf( stderr, "\tdriver error was %s\n", prdvrerr( ( int) regs.h.dh));

                return( ERROR);                                  /* yes, error */
        }

        dvrinfop->version = ( int) regs.x.bx;        /* set the interface version */
        dvrinfop->class = ( int) regs.h.ch;          /* set the interface class */
        dvrinfop->type = ( int) regs.x.dx;           /* set the interface type */
        dvrinfop->number = ( int) regs.h.cl;         /* set the interface number */
        dvrinfop->flag = ( int) regs.h.al;           /* set the basic/exten. flag */
        FP_SEG( dvrinfop->name) = segregs.ds;        /* set the dvr name segment */
        FP_OFF( dvrinfop->name) = regs.x.si;         /* set the dvr name offset */

        return( 0);
}

// END GTDVRINF.C
```

```
/*---- initdvr() ----------------------------------------------------*/
/*  Initialize the packet driver using the software interrupt 'pdvector'.
Given the class, type, and number of the interface using pdvector, send
it the 'type' of packets to receive, the length of the 'type' specification
(typelen), and the address (buffsubr) of the routine which manages
memory buffers for the packet driver.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet driver header */
#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                          /* for int86x() */

int initdvr( struct dvrinfo *dvrinfop, unsigned char *typep, int typelen,
                                                        char far *buffsubr)
{
        extern int pdvector;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        char *prdvrerr( int);
        int getlocaddr( int, unsigned char *, int);
        struct SREGS segregs;
        union REGS regs;
        int handle, status;

        segread( &segregs);                                     /* get the segment registers */
        regs.h.ah = 2;                                          /* set the access_type() function number */
        regs.h.al = ( unsigned int) dvrinfop->class;            /* set the class no. */
        regs.x.bx = ( unsigned int) dvrinfop->type;             /* set the type no. */
        regs.h.dl = ( unsigned int) dvrinfop->number; /* set the number no. */
        regs.x.si = ( unsigned int) typep;                      /* set the 'type' location */
        regs.x.cx = ( unsigned int) typelen;                    /* set the 'type' len */
        segregs.es= ( unsigned int) FP_SEG( buffsubr);          /* set the buffer manager routine's segment location */
        regs.x.di = ( unsigned int) FP_OFF( buffsubr);          /* set the buffer manager routine's offset location */

        int86x( pdvector, &regs, &regs, &segregs);    /* call the software int. */
        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "initdvr: error on int86x call\n");
                fprintf( stderr, "\tdriver error was %s\n",
                prdvrerr( ( int) regs.h.dh));
                return( ERROR);                                 /* yes, error */
        }

        handle = ( int) regs.x.ax;                      /* set the handle */

        if( ( status = getlocaddr( handle, dvrinfop->locethaddr, E_ADD_SIZ)) < 0)
        {
                return( ERROR - 1);
        }
        return( handle);
}

// END INITDVR.C
```

```
/*----- initinfo() ----------------------------------------------*/
/*  From the configuration information in the SNMP database, initialize
the packet driver's local information.  Return zero on success; ERROR
on error.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   "\gary\snmp\snmp_src\snmp\snmp.h"
#include   <stdio.h>

int initinfo( void)
{
        extern struct routinfo *firstrinfop;
        extern struct dvrinfo pdvrinfo;
        int cmpstr( unsigned char *, unsigned char *, int);
        long atonetaddr( char *);
        void free( void *);
        int status;
        struct routinfo *rinfop, *previnfop;
        struct varinfo *vblkp;

        if( firstrinfop == NULL)                    /* is there any database information? */
                return( ERROR);                     /* no, error */

        previnfop = rinfop = firstrinfop;           /* point at the first info block */
        status = cmpstr( rinfop->community, "<Configuration>", 16);     /* is it

        a configuration block? */
        while( ( status != 0) && ( rinfop->nxtblock != NULL))
        {       /* while it isn't, loop */

                previnfop = rinfop;                 /* save the previous block's loc */
                rinfop = rinfop->nxtblock;          /* point at the next */
                status = cmpstr( rinfop->community, "<Configuration>", 16);
        }

        if( status != 0)                            /* was any configuration info found? */
                return( ERROR);                     /* no, error */

        if( rinfop == previnfop)                    /* yes, remove the block from the chain */
                firstrinfop = rinfop->nxtblock;     /* if it was the first, change the pointer to the first block */
        else                                        /* it wasn't the first, change the previous block's pointer */
                previnfop->nxtblock = rinfop->nxtblock;

        vblkp = rinfop->vblockp;                    /* point at the first variable block */
        free( ( void *) rinfop);                    /* release the router info block */
                                                    /* initialize the driver info fields */

        pdvrinfo.locgw[ 0] = pdvrinfo.locipaddr[ 0] = pdvrinfo.locipmask[ 0] = '\0';

        while( vblkp != NULL)
        {                                           /* while there is configuration info, loop */

                int i;
                struct varinfo *prvblkp;
                prvblkp = vblkp;                    /* save the current block's location */
                                                    /* find the '=' in the string */
                for( i = 1; ( i < STRINGSIZ) && ( vblkp->variable[ i] != '='); i++);
                        if( ++i < STRINGSIZ)
                        {                           /* is there more after the '=' sign? */
                                while( ( ( vblkp->variable[ i] == ' ') || ( vblkp->variable[ i] == '\t')) &&
                                                                        ( i < STRINGSIZ))    /* yes */
                                        i++;        /* find the first ascii integer */
                                        if( ( i < STRINGSIZ) && ( ( vblkp->variable[ i] >= '0') ||
                                                        ( vblkp->variable[ i] <= '9')))
                                        {
                                                /* did we find an integer? */

                                                switch( ( vblkp->variable[0] & UCASEMSK))
                                                {       /* yes */
```

```
                                        case( 'G'):  /* if it was a 'g', get the gateway addr */
                                                     if( pdvrinfo.locgw[ 0] == '\0')
                                                      *( ( long *) &( pdvrinfo.locgw[ 0])) =
                                                     atonetaddr( &( vblkp->variable[ i]));
                                                     break;

                                        case( 'I'):  /* if it was an 'i', get the ip mask */
                                                     if( pdvrinfo.locipmask[ 0] == '\0')
                                                     *( ( long *) &( pdvrinfo.locipmask[ 0])) =
                                                     atonetaddr( &( vblkp->variable[ i]));
                                                     break;
                                        case( 'L'):  /* if it was an 'l', get the local ip addr */
                                                     if( pdvrinfo.locipaddr[ 0] == '\0')
                                                     *( ( long *) &( pdvrinfo.locipaddr[ 0])) =
                                                     atonetaddr( &( vblkp->variable[ i]));
                                                }
                                        }       /* else, ignore this block */
                                }       /* else, ignore this block */

                vblkp = vblkp->nxtblock;        /* point to the next block */
                free( ( void *) prvblkp);       /* release the old one */
        }

        if( ( pdvrinfo.locipaddr[ 0] == '\0') || ( pdvrinfo.locgw[ 0] == '\0') || ( pdvrinfo.locipmask[ 0] == '\0'))
                /* were any fields not initialized? */
                return( ERROR);                 /* yes, error */
        return( 0);             /* no errors */
}

// END INITINFO.C
```

```
/*---- getinxstr() ----------------------------------------------*/
/* Given a pointer to a variable info block, a pointer to a character
buffer and its length, put the index in the form of an ASCII string
into the buffer. Return the number of characters put into the
buffer, not including the '\0'. Return ERROR on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for structures */
#include   <stdio.h>              /* for NULL, fprintf(), etc. */

#define    ERROR                  ( -1)

int getinxstr( variable, buffer, buflen)
           struct varinfo *variable;
           char *buffer;
           int buflen;
{
           extern struct routinfo *firstrinfop;
           long decdsubid( unsigned char **);
           struct routinfo *rinfop;
           struct varinfo *vblkp;
           int endflag, i, j;
           unsigned char tempbuf[STRINGSIZ + 16], *bytep, *bytep1;

           endflag = 0;
           rinfop = firstrinfop;    /* make sure 'variable' is a valid pointer */
           while( ( rinfop != NULL) && ( endflag == 0))
           {          /* loop while there are router blocks to look at */

                      vblkp = rinfop->vblockp;          /* point to the first variable block */

                      while( ( vblkp != NULL) && ( endflag == 0))
                      {          /* loop while there are variable blocks to look at */

                                 if( vblkp == variable)  /* is this the right variable block? */
                                            endflag++;              /* yes, set the flag */
                                 else
                                            vblkp = vblkp->nxtblock;          /* no, continue looping */
                      }
                      rinfop = rinfop->nxtblock;          /* go to the next router block */
           }

           if( vblkp != variable)   /* did we find the right variable block? */
                      return( ERROR);                                        /* no, error */

           bytep = ( unsigned char *) &( vblkp->index[0]);
           bytep1 = ( unsigned char *) &( vblkp->index[ ( int) vblkp->indexlen]);
           i = 0;

           while( ( bytep < bytep1) && ( i < STRINGSIZ))
           {
                      sprintf( ( char *) &tempbuf[i], "%d.", ( int) decdsubid( &bytep));

                      while( ( tempbuf[i] != (unsigned char) '\0') && ( i < STRINGSIZ))
                                 i++;
           }

           if( bytep < bytep1)                    /* index too large for temp buffer? */
                      return( ERROR - 1);                    /* yes, error */

           if( i > buflen)                        /* given buffer too small? */
                      return( ERROR - 2);                    /* yes, error */

           if( i > 0)
           {                                      /* was there an index? */
                      tempbuf[i - 1] = '\0';      /* yes, overwrite the last '.' with '\0' */

                      for( j = 0 ; j < i ; j++)   /* move it to the user's buffer */
                                 buffer[j] = tempbuf[j];
           }
```

```
        else
        {                       /* there was no index */
                if( buflen < 2)                 /* user's buffer too small? */
                        return( ERROR - 3);             /* yes, error */

                buffer[ 0] = '0';       /* put zero in user's buffer */
                buffer[ 1] = '\0';      /* null terminate the string */
                i = 1;                  /* set the number of characters in the buffer */
        }
        return( i);
}

// END GETINXST.C
```

```
/*----- getnxtchange() ----------------------------------------*/
/* Given a pointer to a router info block and a pointer to a
variable info block associated with that router, return a
pointer to the next block associated with the same router in which
the change flag is set. If the given pointer to a variable block is
NULL, return the first such block. Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for structures */
#include   <stdio.h>                                  /* for NULL, fprintf(), etc. */

struct varinfo *getnxtchange( router, variable)
        struct routinfo *router;
        struct varinfo *variable;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int endflag, i;

        endflag = 0;
        rinfop = firstrinfop;                              /* make sure 'variable' is a valid pointer */
        while( ( rinfop != NULL) && ( rinfop != router))   /* loop while there are router blocks to look at */
                rinfop = rinfop->nxtblock;                 /* go to the next router block */

        if( rinfop != router)                              /* found the right router block? */
                return( NULL);                             /* no, error */

        if( variable == NULL)
        {                                                  /* asking for first change block? */
                vblkp = rinfop->vblockp;                   /* yes, point to first variable blk */

                if( vblkp->changeflag > 0)                 /* is its change flag set? */
                        return( vblkp);                    /* yes, return a pointer to it */
        }
        else
        {                                                  /* not asking for first block, find given one */
                vblkp = rinfop->vblockp;                   /* point to the first variable block */
                while( ( vblkp != NULL) && ( vblkp != variable))   /* loop while there are variable blocks to look at */
                vblkp = vblkp->nxtblock;                   /* no match, continue looping */

                if( vblkp != variable)                     /* did we find the right variable block? */
                        return( NULL);                     /* no, error */
        }                       /* now we start looking for the next changed block */

        vblkp = vblkp->nxtblock;                           /* point to next block */

        while( ( vblkp != NULL) && ( vblkp->changeflag <= 0))      /* loop while we can't find a changed block */
                vblkp = vblkp->nxtblock;                   /* no match, continue searching */

        if( ( vblkp == NULL) || ( vblkp->changeflag <= 0))  /* found a block? */
                return( NULL);                                      /* no, error */

        return( vblkp);                            /* yes, return a pointer to it */
}

// END GETNXTCH.C
```

```
/*----- getnxtroutp() ----------------------------------------*/
/* Given a pointer to a router info block, find the next router info
block in the chain.  If 'router' is NULL, find the first block in
the chain. Return a the pointer to the next block on success, or
NULL on error.

*/

#include   <stdio.h>   /* for NULL, 'sprintf()', and 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for structures */

struct routinfo *getnxtroutp( router)
struct routinfo *router;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;

        if( router == NULL)                                     /* asking for a pointer to the first block? */
                return( firstrinfop);                           /* yes, return it */

        rinfop = firstrinfop;                                   /* find the current block */

        while( ( rinfop != router) && ( rinfop->nxtblock != NULL))
                        rinfop = rinfop->nxtblock;

        if( rinfop != router)                                   /* found the current block? */
                return( NULL);                                  /* no, error */

        return( router->nxtblock);                              /* return a pointer to the next block */
}

// END GETNXTRP.C
```

```
/*----- getnxtvarp() ----------------------------------------*/
/* Given a pointer to a router info block and a pointer to a
variable info block, return a pointer to the next variable info
block for that router. If the given variable info block is NULL,
return the first variable info block for the given router.  Return
NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"
#include   <stdio.h>                                    /* for NULL, fprintf(), etc. */

struct varinfo *getnxtvarp( router, variable)
        struct routinfo *router;
        struct varinfo *variable;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;

        /* check to see if the router block given is a valid one */

        rinfop = firstrinfop;                                /* point to the first router block */

        /* loop while this router block pointer is not the one we want */

        while( ( rinfop != router) && ( rinfop->nxtblock != NULL))
                rinfop = rinfop->nxtblock;                   /* point to the next block */

        if( rinfop != router)                                /* found the right block? */
                return( NULL);                               /* no, error */

        if( variable == NULL)                        /* yes, is the first variable block desired? */
                return( rinfop->vblockp);                    /* yes, return it */

        vblkp = rinfop->vblockp;                             /* point to the first variable block */
                                                             /* loop while we're not at the right block */

        while( ( vblkp != variable) && ( vblkp->nxtblock != NULL))
                vblkp = vblkp->nxtblock;                     /* point to the next block */

        return( vblkp->nxtblock);          /* return the pointer to the next block, whether there is a block or the pointer is NULL */
}

// END GETNXTVP.C
```

```
/*---- getroutaddr() ----------------------------------------*/
/* Given a pointer to a routing info block, put the address of the
router in the buffer 'buff'.  Return the number of characters, not
including the '\0', or ERROR on error.

*/

#include    "\gary\snmp\snmp_src\snmp\snmp.h"

#define     ERROR               ( -1)

int getroutaddr( router, buff, buflen)
            struct routinfo *router;
            char *buff;
            int buflen;
{
            int i;
            char *bytep;

            bytep = ( char *) &( router->routeraddr[0]);

            for( i = 0 ; ( bytep[i] != '\0') && ( i < buflen) ; i++)
                        buff[i] = bytep[i];

            if( bytep[i] != '\0')
                        return( ERROR);

            buff[i] = bytep[i];                 /* move the '\0' to the buffer */

            return( i);
}

// END GETRADDR.C
```

```
/*---- getrouterp() ----------------------------------------------*/
/* Given a pointer to a string containing the ascii address of a
router and a pointer to a router info block, return a pointer to the
next router info block matching the ascii address after the pointed
to block, or return NULL on error. If the router block pointer is
null, return the first router block matching the address.

*/

#include   <stdio.h>   /* for NULL, 'sprintf()', and 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for structures */

struct routinfo *getrouterp( routeraddr, routinfop)
        char *routeraddr;
        struct routinfo *routinfop;
{
        extern struct routinfo *firstrinfop;
        int cmpstr( unsigned char *, unsigned char *, int);
        struct routinfo *rinfop;
        int i, status, nxtflag;

        rinfop = firstrinfop;                                   /* find the first block */
                                                                /* get the length of the given address string */

        for( i = 0 ; routeraddr[i] != '\0' ; i++);

        i++;                                                    /* add in the '\0' character */

        if( routinfop == NULL)                                  /* return the first block found? */
                nxtflag = 1;                                    /* yes */
        else
                nxtflag = 0;                                    /* no, find a later block */
        do
        {       /* find the block with the matching address string */

                status = cmpstr( ( unsigned char *) routeraddr,  ( unsigned char *) &( rinfop->routeraddr[0]), i);
                if( status != 0)                                /* did the strings match */
                        rinfop = rinfop->nxtblock;              /* no, go to the next block */
                else
                {                                               /* the strings matched */

                        if( nxtflag <= 0)
                        {                                       /* do we want this block? */

                                status = 1;                     /* no, negate the match */
                                if( routinfop == rinfop)        /* do we want the next block? */
                                nxtflag = 1;                    /* yes */
                                rinfop = rinfop->nxtblock;      /* point to the next block */
                        }
                         /* else, we want this block, do nothing */
                }

        } while( ( status != 0) && ( rinfop != NULL));          /* loop while the strings didn't match and there is a next block */

        return( rinfop);                                        /* return either NULL or a pointer to the right block */
}

// END GETROUTP.C
```

```
/*----- getvarchgfl() ----------------------------------------------*/
/* Given a pointer to a variable info block, return the value of the
changeflag.  A positive value indicates a change has taken place, zero
indicates no change, negative values indicate errors.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for structures */
#include   <stdio.h>                                            /* for NULL, fprintf(), etc. */

#define    ERROR              ( -1)

int getvarchfl( variable)
        struct varinfo *variable;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int endflag;

        endflag = 0;
        rinfop = firstrinfop;                                   /* make sure 'variable' is a valid pointer */

        while( ( rinfop != NULL) && ( endflag == 0))
        {loop while there are router blocks to look at */

                vblkp = rinfop->vblockp;                        /* point to the first variable block */

                while( ( vblkp != NULL) && ( endflag == 0))
                {       /* loop while there are variable blocks to look at */

                        if( vblkp == variable)                  /* is this the right variable block? */
                                endflag++;                      /* yes, set the flag */
                        else
                                vblkp = vblkp->nxtblock;        /* no, continue looping */
                }
                rinfop = rinfop->nxtblock;                      /* go to the next router block */
        }

        if( vblkp != variable)  /* did we find the right variable block? */
                return( ERROR);                                 /* no, error */

        return( ( int) vblkp->changeflag);
}

// END GETVARFL.C
```

```
/*---- getvarp() ----------------------------------------------*/
/* Given pointers to strings containing a router address, a variable
name, and an index, return a pointer to a variable info block that
matches the criteria.  Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for structures */
#include   <stdio.h>                                            /* for NULL, fprintf(), etc. */

struct varinfo *getvarp( router, variable, index)
        char *router, *variable, *index;
{
        extern struct objinfo *isoorgroot;
        int cmpstr( unsigned char *, unsigned char *, int);
        struct objinfo *searcharr( struct objinfo *, char *);
        int encdindex( unsigned char *, unsigned char *, int, struct objinfo *);
        struct routinfo *getrouterp( char *, struct routinfo *);
        struct varinfo *getnxtvarp( struct routinfo *, struct varinfo *);
        struct routinfo *rinfop;
        struct objinfo *objinfop;
        struct varinfo *vblkp;
        int varlen, status;
        unsigned char tempbuf[STRINGSIZ];
                                        /* is the router address valid? */

        if( ( rinfop = getrouterp( router, NULL)) == NULL)
        {
                return( NULL);                                  /* no, error */
        }

        if( ( objinfop = searcharr( /* mibroot */ isoorgroot, variable)) == NULL)
        {
                                                /* is 'variable' a MIB variable? */
                return( NULL);                  /* no, error */
        }                                       /* yes it is, get the variable string's length */

        for( varlen = 0 ; variable[varlen] != '\0' ; varlen++);
        varlen++;                               /* include the '\0' */
        vblkp = NULL;                           /* get the first variable block for the given router */

        do
        {               /* check this variable block for the one we want */
                if( ( vblkp = getnxtvarp( rinfop, vblkp)) == NULL)
                {

                        /* is there a variable block to examine? */

                        return( NULL);                  /* no, return NULL */
                }               /* we have a variable block, test it */

                /* does the given variable string match this variable's string? */

                if( ( status = cmpstr( ( unsigned char *) variable, ( unsigned char *) &( vblkp->variable[0]), varlen)) == 0)
                {
                        /* the variable names match, do the indexes? */
                        /* first encode the given index string */

                        if( ( status = encdindex( ( unsigned char *) index, tempbuf,

                        STRINGSIZ, objinfop)) < 0)
                        {       /* did the index string encode properly? */
                                return( NULL);                  /* no, error */
                        }                                       /* the index string encoded properly */

                        if( status = vblkp->indexlen)
                        { /* do the index lengths match? yes, compare the indexes */

                                status = cmpstr( tempbuf, vblkp->index, status);
                        }
                        else    /* the index lengths didn't match */
```

```
                                        status = (-1);
                        }
                        else                    /* the variable strings didn't match */
                                status = (-1);
        } while( status != 0);          /* loop while we haven't found the right variable block */
        return( vblkp);                 /* return a pointer to the variable block */
}

// END GETVARP.C

/*----- getvarstr() ----------------------------------------------*/
/* Given a pointer to a variable info block, a pointer to a character
buffer and its length, put the variable name into the buffer. Return
the number of characters put into the buffer, not including the '\0'.
Return ERROR on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for structures */
#include   <stdio.h>                                            /* for NULL, fprintf(), etc. */

#define    ERROR               ( -1)

int getvarstr( variable, buffer, buflen)
        struct varinfo *variable;
        char *buffer;
        int buflen;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int endflag, i;

        endflag = 0;
        rinfop = firstrinfop;                                   /* make sure 'variable' is a valid pointer */

        while( ( rinfop != NULL) && ( endflag == 0))
        {       /* loop while there are router blocks to look at */

                vblkp = rinfop->vblockp;                        /* point to the first variable block */

                while( ( vblkp != NULL) && ( endflag == 0))
                {       /* loop while there are variable blocks to look at */

                        if( vblkp == variable)                  /* is this the right variable block? */
                                endflag++;                      /* yes, set the flag */
                        else
                                vblkp = vblkp->nxtblock;        /* no, continue looping */
                }
                rinfop = rinfop->nxtblock;                      /* go to the next router block */
        }
        if( vblkp != variable)                                  /* did we find the right variable block? */
        return( ERROR);                                         /* no, error */

        for( i = 0 ; ( vblkp->variable[i] != '\0') && ( i < buflen) ; i++)
                buffer[i] = vblkp->variable[i];

        if( vblkp->variable[i] != '\0')
                return( ERROR - 1);
        buffer[i] = '\0';

        return( i);
}
// END GETVARST.C

/*----- getvartime() ---------------------------------------------*/
/* Given a pointer to a variable info block, return the time stamp of
that variable. Return ERROR on errors.

*/
```

```
#include    "\gary\snmp\snmp_src\snmp\snmp.h"    /* for structures */
#include    <stdio.h>                           /* for NULL, fprintf(), etc. */

#define ERROR       ( -1)

unsigned long getvartime( variable)
        struct varinfo *variable;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int endflag;
        endflag = 0;

        rinfop = firstrinfop;                   /* make sure 'variable' is a valid pointer */

        while( ( rinfop != NULL) && ( endflag == 0))
        {   /* loop while there are router blocks to look at */

                vblkp = rinfop->vblockp;        /* point to the first variable block */

                while( ( vblkp != NULL) && ( endflag == 0))
                {  /* loop while there are variable blocks to look at */

                        if( vblkp == variable)  /* is this the right variable block? */
                                endflag++;      /* yes, set the flag */
                        else
                                vblkp = vblkp->nxtblock;    /* no, continue looping */
                }
                rinfop = rinfop->nxtblock;              /* go to the next router block */
        }
        if( vblkp != variable)                  /* did we find the right variable block? */
                return( ( long) ERROR);         /* no, error */

        return( vblkp->timestamp);
}

// END GETVARTM.C
```

```
/*---- getvartype() ----------------------------------------------*/
/* Given a pointer to a variable info block, return a pointer to a
character string which indicates what the type of the pointer to
variable is.  Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\mib\mib.h"          /* for objinfo structure def */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"              /* for structures */
#include   <stdio.h>                                      /* for NULL, fprintf(), etc. */

char *getvartype( variable)
        struct varinfo *variable;
{
        extern struct routinfo *firstrinfop;
        extern struct objinfo *isoorgroot;

        static char vartype_s[][ 16] =
        { "int", "unsigned int", "char",
          "unsigned char", "char *", "long",
          "unsigned long", "ASCII string",
          "hex string", "integer string"
        } ;

        struct objinfo *searcharr( struct objinfo *, char *);
        int cmpstr( unsigned char *, unsigned char *, int);

        struct objinfo *objinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int i;

        rinfop = firstrinfop;

        while( rinfop != NULL)
        {
                vblkp = rinfop->vblockp;
                while( vblkp != NULL)
                {
                        if( ( objinfop = searcharr( isoorgroot, ( char *) &( vblkp->variable[ 0]))) == NULL)
                        {
                                fprintf( stderr, "getvartype: %s not in the MIB\n",( char *) &( vblkp->variable[ 0]));
                        }

                        for( i = 0; objinfop->objname[ i] != '\0'; i++);

                        if( vblkp == variable)
                        {
                                switch( vblkp->vartype)
                                {
                                        case ( UINTID):
                                                return( vartype_s[ 0]);
                                        case ( UOCTSTRID):
                                                return( vartype_s[ 8]);

                                        case ( IntegerStrID):
                                                return( vartype_s[ 9]);

                                        case ( IpAddrID):
                                                return( vartype_s[ 7]);

                                        case ( GaugeID):
                                                return( vartype_s[ 6]);
                                        default:
                                                return( NULL);
                                }
                        }
                        else
                                vblkp = vblkp->nxtblock;
                }
                rinfop = rinfop->nxtblock;
```

```
        }
        return( NULL);
}

// END GETVARTY.C
```

```
/*----- getvarval() ----------------------------------------------*/
/* Given a pointer to a variable info structure and a pointer to a
memory location, copy the value of the variable into the storage location.
Return the number of bytes used for the value, or ERROR on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for structures */
#include   <stdio.h>                                            /* for NULL, fprintf(), etc. */

#define    ERROR              ( -1)

int getvarval( variable, storage, storsize)
        struct varinfo *variable;
        void *storage;
        int storsize;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int endflag,
        valuelen, i, j;
        unsigned char *bytep;

        endflag = 0;
        rinfop = firstrinfop;                                   /* make sure 'variable' is a valid pointer */

        while( ( rinfop != NULL) && ( endflag == 0))
        {       /* loop while there are router blocks to look at */

                vblkp = rinfop->vblockp;                        /* point to the first variable block */

                while( ( vblkp != NULL) && ( endflag == 0))
                {       /* loop while there arevariable blocks to look at */

                        if( vblkp == variable)                  /* is this the right variable block? */
                                endflag++;                      /* yes, set the flag */
                        else
                                vblkp = vblkp->nxtblock;        /* no, continue looping */
                }
                rinfop = rinfop->nxtblock;                      /* go to the next router block */
        }

        if( vblkp != variable)                                  /* did we find the right variable block? */
                return( ERROR);                                 /* no, error */

        switch( vblkp->vartype)
        {               /* find the size of the value */
                case ( UINTID):
                                valuelen = sizeof( int);
                                bytep = ( unsigned char *) vblkp->varvalue;
                                break;
                case ( UOCTSTRID):
                case ( IntegerStrID):
                case ( DisplStrID):
                case ( UOBJIDID):
                                valuelen = *( ( int *) vblkp->varvalue);
                                bytep = ( ( unsigned char *) vblkp->varvalue + sizeof( int));
                                break;
                case ( IpAddrID):
                                valuelen = 4;
                                bytep = ( unsigned char *) vblkp->varvalue;
                                break;
                case ( CounterID):
                case ( TimeTicksID):
                case ( GaugeID):
                                valuelen = sizeof( unsigned long);
                                bytep = ( unsigned char *) vblkp->varvalue;
                                if( valuelen > storsize)
                                        return( ERROR - 1);
```

```
                    if( ( vblkp->vartype == UOCTSTRID) || ( vblkp->vartype == IpAddrID) ||
                            ( vblkp->vartype == DisplStrID) || ( vblkp->vartype == UOBJIDID) ||
                            ( vblkp->vartype == IntegerStrID))
{
                    /* is the data string data? */
        if( vblkp->vartype == UOCTSTRID)
        {

                if( ( valuelen + valuelen + valuelen + 1) > storsize)
                            /* yes, enough room? */

                return( ERROR - 3);   /* no, error */

                for( i = 0, j = 0 ; i < valuelen ; i++)
                {  /* convert it to ASCII */

                        sprintf( &( ( ( char *) storage)[ j]), "%02x-", bytep[ i]);

                        while( ( ( ( char *) storage)[ j] != '\0') && ( j < storsize))
                                j++;
                }

                ( ( char *) storage)[ j - 1] = '\0';

                valuelen = j;           /* save the length of the converted string */
        }
        else
        {                               /* show it as ASCII data */

                if( ( vblkp->vartype == IpAddrID) ||
                        ( vblkp->vartype == IntegerStrID) ||
                        ( vblkp->vartype == UOBJIDID))
                { /* is it binary? yes, translate it into ASCII and move it */

                        for( i = 0, j = 0 ; ( i < valuelen) && ( j < storsize) ; i++)
                        {
                                sprintf( &( ( ( char *)storage)[ j]), "%d.", ( int) bytep[ i]);

                                while( ( ( ( char *) storage)[ j] != '\0') &&   ( j < storsize))
                                        j++;
                        }

                        if( ( j >= storsize) && ( i < valuelen))
                                /* storage not large enough? */
                                return( ERROR - 7);             /* yes, error */

                        ( ( char *) storage)[ j - 1] = '\0';            /* make the last '.' a '\0' */

                                valuelen = j;
                }
                else
                {       /* it's an ASCII string (DisplStrID), just move it */

                        if( storsize < ( valuelen + 1))                 /* enough room for '\0'? */
                                return( ERROR - 4);                     /* no, error */

                        for ( i = 0 ; i < valuelen ; i++)               /* move it */
                                ( ( char *) storage)[ i] = bytep[ i];

                        ( ( char *) storage)[ i++] = '\0';              /* terminate it */

                        valuelen = i;                                   /* save the length of the moved string */
                }
        }
}
else
{               /* not data string, just move it */

        for( i = 0 ; i < valuelen ; i++)
                ( ( unsigned char *) storage)[ i] = bytep[ i];
```

```
        }

        return( valuelen);
}

// END GETVARVL.C

/*----- initrcv() ----------------------------------------*/
/* Initialize the SNMP monitoring code.  Return ERROR on error; zero
on success.

*/

#include   <stdio.h>            /* for NULL, fprintf(), etc. */

#define    ERROR                ( -1)

int initrcv( void)
{
        int settraps( void);
        int status;

        if( ( status = settraps()) < 0)
        {
                fprintf( stderr, "initrcv: can't set traps, error %d\n", status);
                return( ERROR - 3);
        }
        return( 0);
}

// END INITRCV.C
```

```
/*----- initsnmp() ----------------------------------------------*/
/* Initialize the MIB database and the UDP driver.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for structures */
#include   <stdio.h>                  /* for NULL, fprintf(), etc. */

#define    ERROR                      ( -1)

int initsnmp( char *dbase, char *mibptr[])
{
          extern struct routinfo *firstrinfop;
          int initmib( char *, char **);
          int initudp( void);
          int initrcv( void);
          int status;

          if( ( dbase == NULL) || ( mibptr == NULL))
          {
                    if( dbase == NULL)
                              fprintf( stderr, "initsnmp: SNMP database file is null\n");

                    if( mibptr == NULL)
                              fprintf( stderr, "initsnmp: MIB file pointer is null\n");

                    return( ERROR);
          }

          if( initmib( dbase, mibptr) < 0)
          {         /* initialize the MIB database */

                    fprintf( stderr, "initsnmp: can't initialize MIB database\n");
                    return( ERROR - 1);
          }
          if( firstrinfop == NULL)
          {
                    fprintf( stderr, "initsnmp: no SNMP database info exists\n");
                    return( ERROR - 2);
          }
          if( ( status = initudp()) < 0)
          {
                    fprintf( stderr, "initsnmp: can't initialize UDP driver\n");
                    return( ERROR - 3);
          }
          if( ( status = initrcv( )) < 0)
          {
                    fprintf( stderr, "initsnmp: call to initrcv failed, error %d\n", status);
                    return( ERROR - 4);
          }
          return( 0);
}

// END INITSNMP.C
```

```
/*----- pollagents() ----------------------------------------------*/
/* Check the status of the routers by interrogating them.  Return
ERROR on error; zero on success.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for routinfo struct */
#include   "\gary\snmp\snmp_src\net\net.h"            /* for NOWAITINGFL, sockdata struct */
#include   <stdio.h>          /* for NULL, fprintf(), etc. */

#define    ERROR              ( -1)
#define    ENETPKTSIZ         1500      /* max. data bytes in an Ethernet packet */
#define    IPHDRSIZ20                   /* IP packet header size in bytes */
#define    UDPHDRSIZ          8         /* UDP packet header size in bytes */
#define    REQBUFSIZ          (ENETPKTSIZ - IPHDRSIZ - UDPHDRSIZ)          /* Get Request packet size */

int pollagents()
{
        extern struct routinfo *firstrinfop;
        extern struct sockdata *socketroot;
        extern int univerr;
        extern int verbose;

        long getreq( struct routinfo *, unsigned char *, int *);
        int getudpnd( long, unsigned int, unsigned int, unsigned long);
        int udprelease( int);
        int sendudppkt( int, unsigned char *, int, int);
        int setconnreqid( int, long);
        int rcvpkts( void);
        int cktraps( void);
        int cleartraps( void);
        int setrinfop( int, struct routinfo *);
        int deldata( unsigned char *);
        long atonetaddr( char *);

        unsigned char buffer[REQBUFSIZ], *bytep;
        int buflen, nd, plen, flags, status, i;
        struct routinfo *rinfop;
        struct sockdata *sockdatap;
        long requestid;
        unsigned long fhost;

        if( verbose == 2)
                cprintf( "\r\npollagents: checking routers\n\r");

        flags = univerr = 0;
        rinfop = firstrinfop;

        while( rinfop != NULL)
        {
                int tocnt;
                buflen = REQBUFSIZ;

                if( ( requestid = getreq( rinfop, buffer, &buflen)) < 0L)
                {
                                                        /* req pkt formatted? */
                        cprintf( "pollagents: can't format a %s req pkt\n\r", &( rinfop->routeraddr[0]));
                        if( univerr == -33)
                                cprintf( "\tpacket too large\n\r");
                        else
                                cprintf( "\tuniverr = %d\n\r", univerr);

                        return( ERROR);
                }
                        /* packet was successfully formatted */

                if( verbose == 2)
                {
                        cprintf( "pollagents: packet was formatted as\n\r");
                        for( i = 0 ; i < buflen ; i++)
                        cprintf( "%02x ", buffer[i]);
```

```
            cprintf( " \n\r");
}

fhost = atonetaddr( &( rinfop->routeraddr[0]));

if( verbose == 2)
            cprintf( "pollagents: fhost = %08lx, addr = %s\n\r", fhost, &( rinfop->routeraddr[0]));

                                                /* set router address */
if( ( nd = getudpnd( fhost, ( unsigned int)GETREQPORT, ( unsigned int)0,
                                                                        ( unsigned long)NOWAITINGFL)) < 0)
{
            cprintf("pollagents: can't get nd for sending, error %d\n\r", nd);
            return( ERROR - 1);
}

/* got an nd, set the reqid */

if( verbose == 2)
            cprintf( "pollagents: requestid = %ld\n\r", requestid);

setconnreqid( nd, requestid);
setrinfop( nd, rinfop);

if( verbose == 2)
            cprintf( "pollagents: requestid = %ld\n\r", requestid);

tocnt = 0;

do
{
            if( verbose == 2)
                        cprintf( "pollagents: socketroot = %04x\n\r", socketroot);

                                    /* send the request packet */
                        /* buflen contains the length of the buffer */

            if( ( plen = sendudppkt( nd, buffer, buflen, flags)) < 0)
            {
                        cprintf( "pollagents: can't send a request, error %d\n\r", plen);
                        udprelease( nd);
                        return( ERROR - 2);
            }

                        /* a request has been sent, wait for a reply */

            if( verbose == 2)
            {
                        bytep = ( unsigned char *) &fhost;
                        cprintf( "pollagents: interrogating %d.%d.%d.%d\n\r",     ( int)*bytep, ( int)*( bytep + 1),
                                                                        ( int)*( bytep + 2), ( int)*( bytep +i));

                        cprintf( "\twith request id %ld\n\r", requestid);
            }

            if( ( status = rcvpkts()) < 0)
            {
                        cprintf( "pollagents: error %d on receiving packets\n\r", status);

                        if( status == -5)
                        {
                                    cprintf( "\trequest packet format was bad\n\r");
                                    udprelease( nd);
                                    return( ERROR - 3);
                        }
            }
            tocnt++;

} while( ( tocnt < 5) && ( status < 1));

if( status < 1)
```

```
                {
                        bytep = ( unsigned char *)&fhost;

                        cprintf( "cannot poll %d.%d.%d.%d\n\r", ( int)*bytep, ( int)*( bytep + 1), ( int)*( bytep + 2),
                                                                                                ( int)*( bytep + 3));
                        deldata( ( unsigned char *) &fhost);
                }

                udprelease( nd);                        /* give up on this agent */
                rinfop = rinfop->nxtblock;
        }

        if( verbose == 2)
        cprintf( "pollagents: socketroot = %04x\n\r", socketroot);

        if( ( status = cktraps()) < 0)
        {
                cprintf( "pollagents: error %d on setting traps\n\r", status);
                return( ERROR - 4);
        }

        if( verbose == 2)
                cprintf( "pollagents: socketroot = %04x\n\r", socketroot);

        return( 0);
}

// END POLLAGEN.C
```

```
/*---- prhexstr() ----------------------------------------------*/
/* Given a pointer to a string of bytes, the number of bytes to convert,
the buffer to put them in, and the size of the buffer, interpret the
string as hexadecimal data to be converted to ASCII in the given buffer.
Return the number of bytes used in the buffer, or ERROR on error.

*/

#include   <stdio.h>                /* for NULL, fprintf(), etc. */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for STRINGSIZ */

#define    ERROR                    ( -1)

int prhexstr( hexdata, numbytes, buffer, buflen)
           unsigned char *hexdata;
           char *buffer;
           int numbytes, buflen;
{
           int i, j;
           char tempbuf[STRINGSIZ + 16];

           for( i = 0, j = 0 ; ( i < numbytes) && ( j <= STRINGSIZ) ; i++)
           {
                      sprintf( &tempbuf[j], "%02x", hexdata[i]);

                      while( ( tempbuf[j] != '\0') && ( j <= ( STRINGSIZ + 16)))
                                 j++;
           }

           if( i < numbytes)
                      return( ERROR);

           if( j >= buflen)
                      return( ERROR - 1);

           for( i = 0 ; i <= j ; i++)
                      buffer[i] = tempbuf[i];

           return( j);
}

// END PRHEXSTR.C
```

```
/*----- rcvpkts() ----------------------------------------------*/
/* If there are any packets to process, do so.  If there are replies
expected to transmitted packets, loop for a while waiting for the
replies. If the replies aren't received, stop listening for them and
go back to listening for trap packets.

*/

#include   "\gary\snmp\snmp_src\net\net.h"          /* for structures */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"        /* for TRAPID, etc. */
#include   <stdio.h>                               /* for NULL, fprintf(), etc. */

#define    ENETPKTSIZ      1500                    /* max. bytes in an Ethernet packet */
#define    ERROR           ( -1)

int rcvpkts()
{
        extern int verbose;

        int getudppkt( int, unsigned char *, unsigned int, unsigned int);
        int udprelease( int);
        int getconninfo( int, struct sockdata *);
        int setconnreqid( int, long);
        long getconnreqid( int);
        int analtrap( int, unsigned char *);
        int wait( unsigned int);
        char *checkpkt( unsigned char *, int, int);
        int analresp( int, unsigned char *);
        int getnxtsendnd( int);
        int getnxttrapnd( int);

        struct sockdata sockinfo;
        int nd, status, tocnt, plen, flags, activflag, errflag, buflen, i;
        long requestid;
        unsigned char buffer[ENETPKTSIZ],                   /* received packet buffer */
        *bytep,
        *pdu;                                               /* pdu object pointers */

        errflag = activflag = flags = 0;
        nd = getnxttrapnd( ( int)( -1));                    /* get first trap nd */

        while( nd >= 0)
        {               /* loop while there are trap nd's */
                do
                {
                        if( ( plen = getudppkt( nd, buffer, ( int)ENETPKTSIZ, flags)) > 0)
                        {
                                /* any queued trap packets to process? */

                                if( verbose == 1)
                                {
                                        cprintf( "\n\r\nrcvpkts: the received packet was\n\r");
                                        for( i = 0 ; i < plen ; i++)
                                                cprintf( "%02x ", buffer[i]);

                                        cprintf( " \n\r");
                                }

                                if( ( pdu = ( unsigned char *) checkpkt( buffer, plen, nd))!= NULL)
                                {
                                        /* is the packet an SNMP packet? */

                                        if( *pdu == TRAPID)
                                        {       /* yes, is it a trap packet? */
                                                if( verbose == 1)
                                                {
                                                   bytep = ( unsigned char *)&( sockinfo.fhost);
                                                   cprintf("received a trap packet from  %d.%d.%d.%d\n\r",
                                                        ( int)*bytep, ( int)*( bytep + 1),
                                                        ( int)*( bytep + 2), ( int)*( bytep + 3));
                                                }
```

```
                                        activflag++;

                                        if( ( status = analtrap( nd, pdu)) < 0)
                                        {
                                        /* analyze trap data successful? */

                                          cprintf("rcvpkts: trap packet analysis error %d\n\r", status);
                                          errflag = -1;
                                        }
                                        else
                                        {
                                                if( verbose == 1)
                                                cprintf("rcvpkts: successfully analyzed a trap pkt\n\r");
                                        }

                                }
                                else
                                {       /* non-trap SNMP packet received on trapnd */
                                        cprintf(
                                        "rcvpkts: non-trap SNMP packet received on trapnd = %d\n\r", nd);
                                        errflag = -2;
                                }
                        }
                        else
                        {       /* non-SNMP packet received on trapnd */
                                cprintf("rcvpkts: non-SNMP packet received on trapnd = %d\n\r", nd);
                                errflag = -3;
                        }
                }
                else
                {                       /* plen <= 0 */
                        if( plen < 0)
                        {
                                cprintf("rcvpkts: error %d on trap nd %d\n\r", plen, nd);

                                if( plen == ERROR)
                                        cprintf( "\tno net descriptors allocated\n\r");
                                else
                                {
                                        if( plen == ( ERROR - 1))
                                                cprintf( "\tnet descriptor not allocated\n\r");
                                        else
                                                cprintf("\terror on reading from net driver\n\r");
                                }
                                errflag = -4;
                        }
                }
        } while( plen > 0);     /* loop while there are still packets queued on this nd */

        nd = getnxttrapnd( nd);         /* go to the next trap nd */
}
nd = getnxtsendnd( ( int)( -1));        /* get first send nd */

while( nd >= 0)
{               /* loop while there are send nd's */
        if( ( requestid = getconnreqid( nd)) >= 0L)
        {
                                /* any outstanding replies? */
                tocnt = 0;      /* yes, wait for them */
                do
                {       /* look for queued pkts on this nd */
                        if( ( plen = getudppkt( nd, buffer, ( int)ENETPKTSIZ, flags)) > 0)
                        {
                                if( verbose == 1)
                                {
                                        cprintf( "rcvpkts: the received packet was\n\r");

                                        for( i = 0 ; i < plen ; i++)
                                                cprintf( "%02x ", buffer[i]);

                                        cprintf( " \n\r");
```

```
                }
                if( ( pdu = ( unsigned char *) checkpkt( buffer, plen, nd)) != NULL)
                {
                        /* is the packet an SNMP packet? */

                        if( *pdu == RESPID)
                        {       /* yes, is it a response pkt? */
                                if( verbose == 1)
                                {
                                  bytep = ( unsigned char *)&( sockinfo.fhost);
                                  cprintf(
                                  "received a response packet from .%d.%d.%d\n\r",
                                        ( int)*bytep, ( int)*( bytep + 1),
                                        ( int)*( bytep + 2), ( int)*( bytep + 3));
                                }
                                activflag++;

                                if( (status = analresp( nd, pdu)) < 0)
                                {
                                        /* yes, analysis successful? */
                                cprintf(
                                "rcvpkts: error %d on analyzing the response
                                                packet\n\r", status);
                                        /* error -14 indicates error-status in pkt */
                                        errflag = -5;
                                }
                                else
                                {       /* response pkt analysis successful */
                                if( verbose == 1)
                                cprintf(
                                "rcvpkts:successfully analyzed the response
                                                                packet\n\r");
                                setconnreqid( status, ( long)( -1));
                                                /* clear status of this nd */
                                }
                        }
                        else
                        {                       /* non-response SNMP packet received */
                                cprintf(
                        "rcvpkts: non-response SNMP packet received on nd %d\n\r", nd);
                                plen = 0;
                                tocnt--;
                                errflag = -6;
                        }
                }
                else
                {                       /* non-SNMP packet received */
                        cprintf(
                                "rcvpkts: non-SNMP packet received on nd %d\n\r", nd);
                        plen = 0;
                        tocnt--;
                        errflag = -7;
                 }
        }
        else
        {               /* packet length is <= 0 */
                if( plen < 0)
                {
                        tocnt = 5;              /* unknown error, exit */
                        cprintf(
                        "rcvpkts: error %d received on send nd %d\n\r",
                        plen, nd);
                        if( plen == ERROR)
                                cprintf("\tno net descriptors allocated\n\r");
                        else
                        {
                                if( plen == ( ERROR - 1))
                                        cprintf("\tnet descriptor not allocated\n\r");
                                else
                                    cprintf("\terror on reading from net river\n\r");
                        }
```

```
                                        errflag = -8;
                                    }
                                }                               /* done with this check on the queue */
                                tocnt++;                        /* increment the time-out count */
                                wait( 1000);                    /* wait one thousand milliseconds */
                            } while( ( tocnt < 5) && ( plen <= 0));   /* loop for five
                                                                seconds while a reply is not received on this send nd */
                        }                                       /* done with outstanding replies on this send nd */
                        nd = getnxtsendnd( nd);                 /* go to the next send nd */
                }                                               /* loop while there are send nd's */
                if( errflag < 0)                                /* any errors? */
                        return( errflag);                       /* yes, return < 0 */

                if( activflag > 0)                              /* any valid activity? */
                        return( 1);                             /* yes, return > 0 */

                return( 0);                                     /* no activity, return 0 */
}

// END RCVPKTS.C

/*----- global variable ----------------------------------------*/
#include    <stdio.h>   /* for NULL, 'sprintf()', and 'fprintf()' */

struct routinfo *firstrinfop = NULL;
int verbose = 0;

// END UIGLOB.C
```

```
/*---- xlatevalue() ------------------------------------------------*/
/* Given a pointer to a variable info block, translate the numeric
value of the variable into a character string suitable for sending
to a console. Return a pointer to the character string on success;
NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for structures */
#include   "\gary\snmp\snmp_src\mib\mib.h"            /* for structures */
#include   <stdio.h>              /* for NULL, fprintf(), etc. */

char *xlatevalue( struct varinfo *varinfop)
{
        extern struct objinfo *isoorgroot;

        struct objinfo *searcharr( struct objinfo *, char *);
        int cmpstr( unsigned char *, unsigned char *, int);
        char *getvartype( struct varinfo *);
        int getvarval( struct varinfo *, void *, int);

        struct objinfo *objinfop;
        int i, j;
        unsigned long li, lj;
        unsigned char *ubytep;
        char *bytep;
        static char tempbuf[STRINGSIZ];


        if( ( objinfop = searcharr( /* mibroot */ isoorgroot,
                                        ( char *) &( varinfop->variable[0])) == NULL)
        {
                fprintf( stderr, "xlatevalue: %s isn't a MIB variable\n",
                                                        &( varinfop->variable[0]));
                return( NULL);
        }

        if( objinfop->ptr == NULL)
        {

                return( NULL);
        }

        if( ( ubytep = ( unsigned char *) getvartype( varinfop)) == NULL)
        {
                fprintf( stderr, "xlatevalue: can't get variable type for %s\n", ( char *) &( varinfop->variable[0]));
                return( NULL);
        }

        for( i = 0 ; ubytep[i] != '\0' ; i++);
        i++;

        if( cmpstr( ubytep, ( unsigned char *) "int", i) == 0)
        {
                if( getvarval( varinfop, ( void *) &i, sizeof(i)) < 0)
                {
                        fprintf( stderr, "xlatevalue: can't get an integer value\n");
                        return( NULL);
                }
                for( j = 0 ; ( j < i) && ( ( ( char **) ( objinfop->ptr))[j] != NULL) ; j++);

                if( ( ( ( char **) ( objinfop->ptr))[j] == NULL) || ( i < 0))
                        return( NULL);

                return( ( ( char **)( objinfop->ptr))[i]);
        }
        else
        {
                if( cmpstr( ubytep, ( unsigned char *) "unsigned long", i) == 0)
                {
                        if( getvarval( varinfop, ( void *) &li, sizeof( li)) < 0)
```

```
                {
                        fprintf( stderr, "xlatevalue: can't get a long value\n");
                        return( NULL);
                }
                for ( lj = 0L ; ( lj < li) &&
                        ( ( ( char **) ( objinfop->ptr))[ ( int)lj] != NULL) ;

                                lj++);
                if( ( ( ( char **) ( objinfop->ptr))[ ( int)lj] == NULL) ||

                        ( li < 0L))
                        return( NULL);

                return( ( ( char **) ( objinfop->ptr))[( int) li]);
        }
        else
        {
                if( cmpstr( ubytep, ( unsigned char *) "ASCII string", i) == 0)
                {
                        return( NULL);
                }
                else
                {
                        if( cmpstr( ubytep, ( unsigned char *) "hex string", i) == 0)
                        {
                                return( NULL);
                        }
                        else
                        {
                                fprintf( stderr, "xlatevalue: \"%s\" is currently unsupported\n",

                                ubytep);
                                return( NULL);
                        }
                }
        }
    }
}

// END XLATEVAL.C
```

```
/*----- analtrap() ----------------------------------------------*/
/* Given a trap protocol data unit (pdu), analyze it. Return non-
zero on error, zero on no error.
*/
#include   <stdio.h>                                   /* for fprintf() */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"           /* for struct objptrs */
#include   "\gary\snmp\snmp_src\mib\mib.h"             /* for struct objinfo */

#define    ERROR          (-1)                        /* error returned status */

int analtrap( nd, pdu)
   int nd;
   unsigned char *pdu;
{
   extern struct objinfo traptypes[];
   int getobjptrs( unsigned char *, struct objptrs *);
   struct objinfo *getobjidstrc( unsigned char *);
   unsigned char *getnxtobj( unsigned char *);
   struct objinfo *strcp;
   struct objptrs objp;
   unsigned char *bytep, *endata;
   int i, status;
   long *agentaddr;
                  /* decode the PDU implied sequence */

   if( *pdu != TRAPID)                                 /* pointing to a trap pdu? */
         return( ERROR);                               /* no, return error */

   if( getobjptrs( pdu, &objp) < 0)          /* got object pointers? */
         return( ERROR - 1);                           /* no, error */

   endata = objp.endatap;                              /* find the end of the data */

                                                       /* decode the 'enterprise' variable */

   bytep = objp.datap;                                 /* find the beginning of the data */

   if( *bytep != UOBJIDID)                             /* found the 'enterprise' variable? */
         return( ERROR - 2);                           /* no, error */

   if( getobjptrs( bytep, &objp) < 0)                  /* got object pointers? */
         return( ERROR - 3);                           /* no, error */

   if( objp.endobjp > endata)                          /* moved past the end of the data? */
         return( ERROR);                               /* yes, error */

   bytep = objp.datap;                                 /* point to the data field */

   if( ( bytep = getnxtobj( objp.tagp)) == NULL)       /* found the 'agent- addr' object? */
         return( ERROR - 4);                           /* no, error */

   if( *bytep != IpAddrID)                             /* found the 'agent-addr' variable? */
         return( ERROR - 5);                           /* no, error */

                  /* decode the 'agent-addr' variable */

   if( getobjptrs( bytep, &objp) < 0)                  /* got the 'agent-addr' pointers? */
         return( ERROR - 6);                           /* no, error */

   if( objp.endobjp > endata)                          /* moved past the end of the data? */
         return( ERROR - 7);                           /* yes, error */

   bytep = objp.datap;

   agentaddr = ( long *) bytep;                        /* save a pointer to the agent address */

   if( ( bytep = getnxtobj( objp.tagp)) == NULL)       /* found 'generic-trap' object? */
         return( ERROR - 8);                           /* no, error */

   if( *bytep != UINTID)
   {     /* found the 'generic-trap' variable? */
```

```
            fprintf( stderr, "analtrap: *bytep = %02x\n", *bytep);
            return( ERROR - 9);                         /* no, error */
    }
                        /* decode the 'generic-trap' variable */
    if( getobjptrs( bytep, &objp) < 0)                  /* got 'generic-trap' pointers? */
            return( ERROR - 10);                        /* no, error */

    if( objp.endobjp > endata)                          /* moved past the end of the data? */
            return( ERROR - 11);                        /* yes, error */

    bytep = objp.datap;                                 /* point to the data field */

    if( ( ( int)( *bytep) < 0) || ( ( int)( *bytep) > 6))   /* is it 0 - 6? */
            return( ERROR - 12);                        /* no, error */

    if( ( status = ( traptypes[( int)( *bytep)].funct)( *agentaddr, pdu)) < 0)
                                    /* called the appropriate function successfully? */

            fprintf( stderr, "analtrap: trap function \"%s\"returned error %d\n", traptypes[( int)*bytep].objname, status);

    if( ( bytep = getnxtobj( objp.tagp)) == NULL)       /* found the 'specific-trap' object? */
            return( ERROR - 13);                        /* no, error */

    if( *bytep != UINTID)                               /* found the 'specific-trap' object? */
            return( ERROR - 14);                        /* no, error */

    if( getobjptrs( bytep, &objp) < 0)                  /* got the 'specific-trap' pointers? */
            return( ERROR - 15);                        /* no, error */

    if( objp.endobjp > endata)                          /* moved past the end of the data? */
            return( ERROR - 16);                        /* yes, error */

    bytep = objp.datap;                                 /* point to the data */

    if( ( bytep = getnxtobj( objp.tagp)) == NULL)       /* found 'time-stamp'? */
            return( ERROR - 17);                        /* no, error */

    if( *bytep != TimeTicksID)                          /* found the 'time-stamp' variable? */
            return( ERROR - 18);                        /* no, error */

    if( getobjptrs( bytep, &objp) < 0)                  /* got 'time-stamp' pointers? */
            return( ERROR - 19);                        /* no, error */

    if( objp.endobjp > endata)                          /* moved past the end of the data? */
            return( ERROR - 20);                        /* yes, error */

    bytep = objp.datap;                                 /* point to the data */

    if( ( bytep = getnxtobj( objp.tagp)) == NULL)       /* found 'var-bind'? */
            return( ERROR - 21);                        /* no, error */

    if( *bytep != USEQID)                               /* found the variable? */
            return( ERROR - 22);                        /* no, error */

    if( getobjptrs( bytep, &objp) < 0)                  /* got 'variable-bindings' pointers? */
            return( ERROR - 23);                        /* no, error */

    if( objp.endobjp > endata)                          /* moved past the end of the data? */
            return( ERROR - 24);                        /* yes, error */

    bytep = objp.datap;                                 /* point to the data */
    return( 0);
}
// END ANALTRAP.C

/*----- authenticationFailure trap ----------------------------------*/
#include    <stdio.h>   /* for fprintf() */

int authenticationFailure_f( agentaddr, pdutag)
    long agentaddr;
```

```
  unsigned char *pdutag;
{
  unsigned char *bytep;                                   /* print a message */
  int i;
  bytep = ( unsigned char *) &agentaddr;

  fprintf( stderr, " Agent address ");

  for( i = 0 ; i < sizeof( agentaddr) ; i++)
          fprintf( stderr, "%d.", ( int)*( bytep++));

  fprintf( stdout, " had an authentication failure\n");

  return( 0);
}

// ENDAUTHFAIL.C

/*---- cktraps() ----------------------------------------------*/
/* Check for trap packet reception enabled.  If not, do so.

*/

#include   <stdio.h>  /* for fprintf() */

#define    ERROR              ( -1)

int cktraps()
{
  int getnxttrapnd( int);
  int getnxtsendnd( int);
  int settraps( void);
  int udprelease( int);
  int nd;
  if( ( nd = getnxttrapnd( ( int)( -1))) < 0)
  {       /* are traps set? */

          if( ( nd = settraps()) < 0)
          {           /* can we set traps? */
            fprintf( stderr, "cktraps: can't set traps, error %d\n", nd);
            return( ERROR);                   /* no, error */
          }
  }
  else
  {                                 /* traps are set */
          while( nd >= 0)
          {           /* while more than one trap nd is active */
            if( ( nd = getnxttrapnd( nd)) >= 0)                  /* get the extras */
                    udprelease( nd);                             /* and release them */
          }
  }
  return( 0);                                   /* traps are set */
}

// END CKTRAPS.C
```

```
/*----- cleartraps() ----------------------------------------*/
/* Clear any trap nd's that have been allocated.  Return zero.

*/

int cleartraps()
{
   int getnxttrapnd( int);
   int udprelease( int);
   int nd1, nd2;

   nd1 = getnxttrapnd( ( int)( -1));

   while( nd1 >= 0)
   {
            nd2 = getnxttrapnd( nd1);
            udprelease( nd1);
            nd1 = nd2;
   }

   return( 0);
}

// END CLRTRAPS.C

/*----- coldStart trap ----------------------------------------*/
#include      <stdio.h>      /* for fprintf() */

int coldStart_f( agentaddr, pdutag)
   long agentaddr;
   unsigned char *pdutag;
{
   unsigned char *bytep;                  /* print a message */
   int i;

   bytep = ( unsigned char *) &agentaddr;

   fprintf( stderr, " Agent address ");

   for( i = 0 ; i < sizeof( agentaddr) ; i++)
            fprintf( stderr, "%d.", ( int)*( bytep++));

   fprintf( stdout, " had a cold start\n");

   return( 0);
}

// END COLDSTRT.C

/*----- egpNeighborLoss trap ----------------------------------------*/
#include    <stdio.h>   /* for fprintf() */

int egpNeighborLoss_f( agentaddr, pdutag)
   long agentaddr;
   unsigned char *pdutag;
{
   unsigned char *bytep;                                    /* print a message */
   int i;
   bytep = ( unsigned char *) &agentaddr;
   fprintf( stderr, " Agent address ");

   for( i = 0 ; i < sizeof( agentaddr) ; i++)
            fprintf( stderr, "%d.", ( int)*( bytep++));

   fprintf( stdout, " had an EGP neighbor loss\n");
   return( 0);
}
// END EGPLOSS.C
/*----- enterpriseSpecific trap ----------------------------------------*/
#include    <stdio.h>   /* for fprintf() */
```

```
int enterpriseSpecific_f( agentaddr, pdutag)
   long agentaddr;
   unsigned char *pdutag;
{
   unsigned char *bytep;                              /* print a message */
   int i;
   bytep = ( unsigned char *) &agentaddr;

   fprintf( stderr, " Agent address ");

   for( i = 0 ; i < sizeof( agentaddr) ; i++)
            fprintf( stderr, "%d.", ( int)*( bytep++));

   fprintf( stdout, " had an enterprise specific trap\n");

   return( 0);
}

// END ENTERSPF.C
```

```
/*---- linkDown trap ----------------------------------------*/
/* Given a network descriptor telling which connection the packet
came in on and the protocol data unit (pdu) itself, process the info
for a 'linkDown' trap.  Return ERROR on error.

*/

#include   <stdio.h>                                    /* for 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"            /* for struct routinfo, objptrs */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for struct objinfo */

#define    ERROR          (-1)                         /* error returned status */

int linkDown_f( agentaddr, pdu)
  long agentaddr;
  unsigned char *pdu;
{
  extern unsigned char updnval[];
  extern int univerr;
  extern struct objinfo *isoorgroot;

  unsigned char *getnxtobj( unsigned char *);
  int getobjptrs( unsigned char *, struct objptrs *);
  struct routinfo *getroutinfo( long, struct routinfo *);
  struct varinfo *getvarinfo( struct routinfo *, struct objinfo *, unsigned char *, int);
  int saveval( struct varinfo *, unsigned char *);
  struct objinfo *getobjidstrc( unsigned char *);
  int cmpstr( unsigned char *, unsigned char *, int);

  struct objinfo *searcharr( struct objinfo *, char *);
  struct objptrs objp;
  struct objinfo *objinfop;
  struct routinfo *rinfop;
  struct varinfo *vinfop;
  unsigned char *seqp;
  int linknump, i;

  if( getobjptrs( pdu, &objp) < 0)
  {       /* got the pdu pointers? */

          fprintf( stderr, "linkDown: couldn't get pdu pointers\n");

          return( ERROR);                              /* no, error */
                              /* objp.datap points to the 'Enterprise' variable */
  }

  for( i = 0 ; i < 5 ; i++)                    /* find the 'variable bindings' */
          if( ( objp.datap = getnxtobj( objp.datap)) == NULL)
    {
                                                       /* found the next obj? */
              fprintf( stderr, "linkDown: couldn't skip to variable bindings\n");
              return( ERROR - 1);                      /* no, error */
            }

    if( getobjptrs( objp.datap, &objp) < 0)
    {     /* found the variable bindings sequence? */

              fprintf( stderr, "linkDown: couldn't find var bind sequence pointers\n");
              return( ERROR - 2);                      /* no, error */
    }

    if( getobjptrs( objp.datap, &objp) < 0)
    {     /* found obj id sequence? */

              fprintf( stderr, "linkDown: couldn't find obj id sequence pointers\n");
              return( ERROR - 3);
    }

    seqp = objp.tagp;              /* save the sequence pointer */

    if( getobjptrs( objp.datap, &objp) < 0)
```

```
{       /* found the obj id pointers? */

            fprintf( stderr, "linkDown: couldn't get obj identifier pointers\n");
            return( ERROR - 4);                                 /* no, error */
}

if( ( objinfop = getobjidstrc( objp.tagp)) == NULL)
{
                                                                /* found object structure? */
            unsigned char *bytep;
            fprintf( stderr, "linkDown: couldn't get obj id structure\n");
            fprintf( stderr, "linkDown: pointers are %04x %04x %04x %04x %04x\n",
                        objp.tagp, objp.lenp, objp.datap, objp.endatap, objp.endobjp);

  bytep = objp.tagp;

            fprintf( stderr, "linkDown: object is\n\t");

            while( bytep < objp.endobjp)
              fprintf( stderr, "%02x ", *( bytep++));

            putc( '\n', stderr);
            return( ERROR - 5);                         /* no, error */
}

if( ( cmpstr( ( unsigned char *) objinfop->objname, ( unsigned char *) "ifIndex", 8)) != 0)
{
                                        /* found the right structure? */
            fprintf( stderr, "linkDown: wrong structure retrieved\n");
            fprintf( stderr, "linkDown: retrieved structure was %s\n", objinfop->objname);
            return( ERROR - 6);                         /* no, error */
}

if( objinfop->funct != NULL)
{       /* ifIndex_f pointer exists? */

        if( ( linknump = ( objinfop->funct)( seqp)) < 0)
        {           /* yes, got  the interface value from ifIndex_f? */

            fprintf( stderr, "linkDown: couldn't get interface value\n");
            return( ERROR - 7);                         /* no, error */
        }

        if( ( objinfop = searcharr( isoorgroot, "ifOperStatus")) == NULL)
        {
            fprintf( stderr, "linkDown: couldn't find ifOperStatus structure\n");
            return( ERROR - 8);
        }
                            /* now update variable database */
        if( ( rinfop = getroutinfo( agentaddr, NULL)) == NULL)
        {
                            /* got router info pointer from the net desc? */
            unsigned char *bytep;                       /* no, print a message */

            bytep = ( unsigned char *) &agentaddr;
            fprintf( stderr, " Agent address ");
            for( i = 0 ; i < sizeof( agentaddr) ; i++)
                    fprintf( stderr, "%d.", ( int)*( bytep++));

            fprintf( stdout, " link %d is down\n", linknump);
            return( 0);                         /* no router info, so print a message */
        }
        else
        {
            if( ( vinfop = getvarinfo( rinfop, objinfop, ( unsigned char *) &linknump, 1)) == NULL)
            {
                                        /* got variable info block? */
                    unsigned char *bytep;                               /* no, print a message */
                    bytep = ( unsigned char *) &agentaddr;
                    fprintf( stderr, " Agent address ");
```

```
                        for( i = 0 ; i < sizeof( agentaddr) ; i++)
                            fprintf( stderr, "%d.", ( int)*( bytep++));

                        fprintf( stdout, " link %d is down\n", linknump);
                        return( 0);         /* no variable info, so print a message */
                }
                updnval[2] = 2;   /* set 'down' value in fake object for saveval() */

                if( ( i = saveval( vinfop, ( unsigned char *) updnval)) < 0)
                {
                        fprintf( stderr, "linkDown: saveval() returned %d\n", i);
                        return( ERROR - 10);
                }
        }
  }
  else                          /* no ifIndex_f function pointer exists */
        return( ERROR - 11);

  return( 0);
}

// END LINKDOWN.C
```

```
/*---- linkUp trap ----------------------------------------------*/
/* Given a network descriptor telling which connection the packet
came in on and the protocol data unit (pdu) itself, process the info
for a 'linkUp' trap.  Return ERROR on error.

*/

#include   <stdio.h>                                   /* for 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"           /* for struct routinfo, objptrs */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for struct objinfo */

#define    ERROR              (-1)                    /* error returned status */

int linkUp_f( agentaddr, pdu)
  long agentaddr;
  unsigned char *pdu;
{
  extern unsigned char updnval[];
  extern int univerr;
  extern struct objinfo *isoorgroot;

  unsigned char *getnxtobj( unsigned char *);
  int getobjptrs( unsigned char *, struct objptrs *);
  struct routinfo *getroutinfo( long, struct routinfo *);
  struct varinfo *getvarinfo( struct routinfo *, struct objinfo *, unsigned char *, int);
  int saveval( struct varinfo *, unsigned char *);
  struct objinfo *getobjidstrc( unsigned char *);
  int cmpstr( unsigned char *, unsigned char *, int);
  struct objinfo *searcharr( struct objinfo *, char *);

  struct objptrs objp;
  struct objinfo *objinfop;
  struct routinfo *rinfop;
  struct varinfo *vinfop;
  unsigned char *seqp;
  int linknump, i;

  if( getobjptrs( pdu, &objp) < 0)
  {       /* got the pdu pointers? */

          fprintf( stderr, "linkUp: couldn't get pdu pointers\n");
          return( ERROR);                             /* no, error */
                  /* objp.datap points to the 'Enterprise' variable */
  }

  for( i = 0 ; i < 5 ; i++)                   /* find the 'variable bindings' */
          if( ( objp.datap = getnxtobj( objp.datap)) == NULL)
          {
                                                      /* found the next obj? */
              fprintf( stderr, "linkUp: couldn't skip to variable bindings\n");
              return( ERROR - 1);                     /* no, error */
          }

          if( getobjptrs( objp.datap, &objp) < 0)
          {           /* found the variable bindings sequence? */

              fprintf( stderr, "linkUp: couldn't find var bind sequence pointers\n");
              return( ERROR - 2);                     /* no, error */
          }

          if( getobjptrs( objp.datap, &objp) < 0)
          {           /* found obj id sequence? */

              fprintf( stderr, "linkUp: couldn't find obj id sequence pointers\n");
              return( ERROR - 3);
          }

          seqp = objp.tagp;                   /* save the sequence pointer */

  if( getobjptrs( objp.datap, &objp) < 0)
  {       /* found the obj id pointers? */
```

```
            fprintf( stderr, "linkUp: couldn't get obj identifier pointers\n");
            return( ERROR - 4);                                 /* no, error */
}

if( ( objinfop = getobjidstrc( objp.tagp)) == NULL)
{       /* found object structure? */

        unsigned char *bytep;
        fprintf( stderr, "linkUp: couldn't get obj id structure\n");
        fprintf( stderr, "linkUp: pointers are %04x %04x %04x %04x %04x\n",
              objp.tagp, objp.lenp, objp.datap, objp.endatap, objp.endobjp);

        bytep = objp.tagp;
        fprintf( stderr, "linkUp: object is\n\t");

        while( bytep < objp.endobjp)
                fprintf( stderr, "%02x ", *( bytep++));

        putc( '\n', stderr);

        return( ERROR - 5);             /* no, error */
  }

  if( ( cmpstr( ( unsigned char *) objinfop->objname, ( unsigned char *) "ifIndex", 8)) != 0)
  {
                                        /* found the right structure? */
            fprintf( stderr, "linkUp: wrong structure retrieved\n");
            fprintf( stderr, "linkUp: retrieved structure was %s\n", objinfop->objname);
            return( ERROR - 6);                         /* no, error */
  }

  if( objinfop->funct != NULL)
  {     /* is there an ifIndex_f pointer? */

        if( ( linknump = ( objinfop->funct)( seqp)) < 0)
        {
        /* yes, gotthe interface index value? */
                fprintf( stderr, "linkUp: couldn't get interface value\n");
                return( ERROR - 7);             /* no, error */
        }


        if( ( objinfop = searcharr( isoorgroot, "ifOperStatus")) == NULL)
        {
                fprintf( stderr, "linkUp: couldn't get ifOperStatus structure\n");
                return( ERROR - 8);
        }
                        /* now update variable database */

        if( ( rinfop = getroutinfo( agentaddr, NULL)) == NULL)
        {
                        /* got router info pointer from the net desc? */
                unsigned char *bytep;           /* no, print a message */

                bytep = ( unsigned char *) &agentaddr;
                fprintf( stderr, " Agent address ");

                for( i = 0 ; i < sizeof( agentaddr) ; i++)
                        fprintf( stderr, "%d.", ( int)*( bytep++));

                fprintf( stdout, " link %d is up\n", linknump);
                return(0);              /* no router info, so print a message */
        }
        else
        {
                if( ( vinfop = getvarinfo( rinfop, objinfop, ( unsigned char *) &linknump, 1)) == NULL)
                {
                                /* got variable info block? */
                        unsigned char *bytep;           /* no, print a message */
```

```
                        bytep = ( unsigned char *) &agentaddr;
                        fprintf( stderr, " Agent address ");

                        for( i = 0 ; i < sizeof( agentaddr) ; i++)
                                fprintf( stderr, "%d.", ( int)*( bytep++));

                        fprintf( stdout, " link %d is up\n", linknump);

                        return( 0); /* no variable info, so print a message */
                }

                updnval[2] = 1;         /* set 'up' value in fake object for saveval() */

                if( ( i = saveval( vinfop, ( unsigned char *) updnval)) < 0)
                {
                        fprintf( stderr, "linkUp: saveval() returned %d\n", i);
                        return( ERROR - 10);
                }
        }
  }
  else                  /* no ifIndex_f pointer exists */

        return( ERROR - 11);

  return( 0);
}

// END LINKUP.C
```

```
/*----- settraps() ----------------------------------------------*/
/* Start the UDP server to listen for SNMP Trap packets.  Return ERROR
on error; zero on success.

*/

#include   <stdio.h>                              /* for fprintf() */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"      /* for TRAPPORT, etc. */
#include   "\gary\snmp\snmp_src\net\net.h"        /* for NOWAITINGFL */

#define    ERROR              ( -1)

int settraps()
{
  int udplisten( long, unsigned int, unsigned int, unsigned long);
  int udprelease( int);
  int setconnreqid( int, long);
  int trapnd;

                             /* set option to non-blocking */
                             /* try to start the UDP server */

  if( ( trapnd = udplisten( 0L, ( int)TRAPPORT, ( int)TRAPPORT,
                                        ( unsigned long)NOWAITINGFL)) < 0)
  {
                  /* listen for any host's trap port on our trap port, error? */
          fprintf( stderr, "settraps: udplisten returned error %d\n", trapnd);
          return( ERROR);                                          /* error */
  }

  if( setconnreqid( trapnd, ( long)( -2)) < 0)
  {       /* marked this nd as a trap, not a send or receive nd? */

          fprintf( stderr, "settraps: couldn't set trapnd %d req id to -2\n", trapnd);
          udprelease( trapnd);
          return( ERROR - 1);                      /* no, error */
  }
  return( 0);
}
// END SETTRAPS.C

/*----- traptypes -----------------------------------------------*/
#include   <stdio.h>  /* for NULL */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for struct objinfo */
/*----- trap name strings -----*/
char traptypes_s[][25] = { "coldStart", "warmStart", "linkDown", "linkUp",
                           "authenticationFailure", "egpNeighborLoss",
                           "enterpriseSpecific"
                         };
/*----- trap structures of pointers -----*/

int coldStart_f( long, unsigned char *);
int warmStart_f( long, unsigned char *);
int linkDown_f( long, unsigned char *);
int linkUp_f( long, unsigned char *);
int authenticationFailure_f( long, unsigned char *);
int egpNeighborLoss_f( long, unsigned char *);
int enterpriseSpecific_f( long, unsigned char *);

struct objinfo traptypes[] = { { traptypes_s[0], 0, NULL, -1, coldStart_f},
                               { traptypes_s[1], 0, NULL, -1, warmStart_f},
                               { traptypes_s[2], 0, NULL, -1, linkDown_f},
                               { traptypes_s[3], 0, NULL, -1, linkUp_f},
                               { traptypes_s[4], 0, NULL, -1, authenticationFailure_f},
                               { traptypes_s[5], 0, NULL, -1, egpNeighborLoss_f},
                               { traptypes_s[6], 0, NULL, -1, enterpriseSpecific_f}
                             };
// END TRAPTYPE.C
/*----- linkUp/linkDown object for saveval() --------------------*/
#include   "\gary\snmp\snmp_src\snmp\snmp.h"      /* for UINTID */
```

```
unsigned char updnval[] = { ( unsigned char) UINTID, 0x01, 0 };


// END UPDNVAL.C

/*----- warmStart trap ----------------------------------------*/
#include   <stdio.h>  /* for fprintf() */

int warmStart_f( agentaddr, pdutag)
  long agentaddr;
  unsigned char *pdutag;
{
  unsigned char *bytep;                                  /* print a message */
  int i;
  bytep = ( unsigned char *) &agentaddr;

  fprintf( stderr, " Agent address ");

  for( i = 0 ; i < sizeof( agentaddr) ; i++)
            fprintf( stderr, "%d.", ( int)*( bytep++));

  fprintf( stdout, " had a warm start\n");

  return( 0);
}

// END WARMSTRT.C

/*----- cmpstr() ----------------------------------------------*/
/* Given pointers to two strings, compare them for the given number
of bytes. If they compare, return zero.

*/

#define    ERROR              ( -1)

int cmpstr( unsigned char *src, unsigned char *dest, int count)
{
  int i;

  if( count <= 0)
            return( ERROR);

  for( i = 0; i < count; i++)
  {
            if( src[ i] != dest[ i])
            return( ERROR);
  }
  return( 0);
}

// END CMPSTR.C
```

```
/*----- fcmpstr() ----------------------------------------------*/
/* Compare 'bytecnt' bytes of two byte strings pointed to by far pointers.
*/

#include   <stdio.h>              /* for fprintf(), etc. */

#define    ERROR                  (-1)

int fcmpstr( unsigned char far *src1, unsigned char far *src2, int bytecnt)
{
  int i;

  if( bytecnt <= 0)
  {                    /* negative or zero byte count? */
          fprintf( stderr, "fcmpstr: bytecnt is %d, error\n", bytecnt);
          return( ERROR - 1);                          /* yes, error */
  }

    for( i = 0 ; i < bytecnt ; i++, src1++, src2++)
    {

          if( *src1 != *src2)          /* non-matching bytes? */
            i = bytecnt + 1;           /* yes, terminate the comparison */
    }

    if( i > bytecnt)                   /* comparison abnormally terminated? */
          return( ERROR);                          /* yes, error */

    return( 0);
}

// END FCMPSTR.C
```

```
/*----- findstr() ------------------------------------*/
/*  A function to find a null terminated string (str) in another
null terminated text string (text). The nulls don't have to match.
Return the location of the first matching character in 'text'.

*/

#include   <stdio.h>                    /* for 'fprintf()', 'getchar()' */
#include   <malloc.h>                   /* for 'malloc()' */

#include   <stdlib.h>                   /* for 'free()' */

unsigned char *findstr( str, text)
     unsigned char *str,
     *text;
{

  void *malloc();

  void free();

  struct shiftblk
  {
            int value;
            struct shiftblk *nxtptr;
  } *( shiftbk[256]),                   /* array of pointers */
           *shiftval,
           *shiftptr;

  int i, index, strlen;
  unsigned char *strend, *strptr, *endptr, *textptr, *textend;

                                /* find the end of the text */
  for( textend = text ; *textend != '\0' ; textend++);

                                /* find the length of the test string */
  for( strend = str ; ( ( int)( strend - str) < 32766) && ( *strend != '\0') ; strend++);

  strlen = ( int)( strend - str);
  shiftval = ( struct shiftblk *)malloc( strlen * sizeof( struct shiftblk));

  if( ( int)( textend - text) < strlen)

          return( NULL);                /* return an error if the text is shorter than the test string */
                                        /* initialize the shift table with the initial value of  the test string length */

  for( i = 0 ; i < 256 ; i++)           /* store in the shift table the shift values for the char in the test string */
             shiftbk[i] = NULL;

  for( strptr = strend - 1, index = 0 ; strptr >= str ; strptr--, index++)
  {
          i = *strptr;
          if( shiftbk[i] == NULL)
               shiftptr = shiftbk[i] = &( shiftval[index]);
          else
          {
                    shiftptr = shiftbk[i];

                    while( shiftptr->nxtptr != NULL)
                              shiftptr = shiftptr->nxtptr;

                    shiftptr = shiftptr->nxtptr = &( shiftval[index]);
          }
          shiftptr->nxtptr = NULL;
          shiftptr->value = ( int)( strend - strptr - 1);

                    /* finished setting up the shift table, begin testing */
  }
 endptr = text + strlen - 1;            /* mark position in text where match testing  begins */

  while( ( textptr = endptr) < textend)
```

```
{  /* while we don't point past the end of text, try to find a match; mark the point in the text to begin testing for matches */

        strptr = strend - 1;    /* point to the end of the string to be matched */

        while( ( *strptr == *textptr) && ( strptr >= str))
        {
                        /* while chars match and the whole string hasn't been tested, loop */

                strptr--;
                textptr--;
                endptr--;
        }

        if( strptr >= str)
        {       /* character mismatch? */

                i = ( int)*( textptr++);
                if( shiftbk[i] == NULL)
                {
                        endptr += strlen;       /* yes, get the shift value and shift */
                }

                else
                {
                        shiftptr = shiftbk[i];

                        while( ( shiftptr->value <= ( strend - strptr - 1)) && ( shiftptr != NULL))
                        {
                                shiftptr = shiftptr->nxtptr;
                        }

                        if( shiftptr == NULL)
                        {
                                endptr += strlen;
                        }
                        else
                        {
                                endptr += shiftptr->value;
                        }
                }
        }
        else
        {
            free( shiftval);
            return( ++textptr);                 /* no, return the pointer to the
                                                                        match location */
         }

  }             /* loop while we don't point past the end of text */

  free( shiftval);

  return( NULL);    /* no match, return an error */
}

// END FINDSTR.C
```

```
/*----- fmovestr() ------------------------------------------------*/
/*  Move a string of characters from one far location to another.
Return the number of characters moved.

*/

#include   <dos.h>              /* for FP_OFF(), etc. */

#define    ERROR                ( -1)

int fmovestr( unsigned char far *src, unsigned char far *dest, int count)
{
  int i;

  if( count <= 0)
          return( ERROR);

  if( FP_SEG( src) < FP_SEG( dest))
  {
          for( i = count - 1; i >= 0; i--)
             dest[ i] = src[ i];
  }
  else
  {
         if( FP_SEG( src) == FP_SEG( dest))
         {
                 if( FP_OFF( src) < FP_OFF( dest))
                 {
                         for( i = count - 1; i >= 0; i--)
                                 dest[ i] = src[ i];
                 }
                 else
                 {
                         for( i = 0; i < count; i++)
                                 dest[ i] = src[ i];
                 }
         }
         else
         {
                 for( i = 0; i < count; i++)
                         dest[ i] = src[ i];
         }
  }

  return( i);
}

// END FMOVESTR.C
```

```
/*----- getreply() ----------------------------------------------*/
/* Test to see if the first character received from the console
matches the one passed to this routine. Discard the rest of the
input line, including the '\n'.

*/

#include   <stdio.h>                    /* for 'getchar()' */

#define    ERROR            (-1)       /* error returned status */
#define    UCASEMSK         0x5f       /* mask converting char to upper case */

int getreply( newchar)
  char newchar;
{
  int reply, tmp;

  if( ( getchar() & UCASEMSK) == ( newchar & UCASEMSK))
            reply = 0;
  else
            reply = 1;

  while( ( ( tmp = getchar()) != '\n') && ( tmp != ERROR));

  if( tmp == ERROR)
            return( ERROR);

  return( reply);
}

// END GETREPLY.C

/*----- movenstr() ----------------------------------------------*/
/* Given a pointer to a null terminated string and a pointer to a
location to which that string is to be replicated, move all of the
bytes including the null terminator. Return the number of bytes
moved on success not including the null terminator; ERROR on failure.

*/

int movenstr( srcp, dstp)
  unsigned char *srcp, *dstp;
{
  int len;

  if( srcp < dstp)
  {                 /* are we moving from a higher location in memory to a lower one? */

            for( len = 0 ; *srcp != '\0' ; len++)          /* moving from higher to lower */
               *( dstp++) = *( srcp++);     /* loop while the byte is not null */

            *dstp = *srcp;                                 /* move the '\0' */
  }
  else
  {                 /* moving from a lower location to a higher one */

            unsigned char *sourcep, *destp;

            for( len = 0, sourcep = srcp ; *sourcep != '\0' ; len++, sourcep++ );
                                       /* find the length and end of the source string */
                      destp = dstp + len;     /* initialize the destination pointer */

            while( sourcep >= srcp)          /* loop while not all bytes have been moved */
                *( destp--) = *( sourcep--);
  }
  return( len);
}
// END MOVENSTR.C
/*----- movestr() ----------------------------------------------*/
/*  Given a pointer to a string of bytes, a pointer to the desired
location of those bytes, and the number of bytes in the string, move
```

them so that the destination string does not overwrite the source
string.

*/

```
#define     ERROR               ( -1)

int movestr( unsigned char *src, unsigned char *dest, int count)
{
   int i;

   if( count <= 0)
            return( ERROR);

   if( src < dest)
   {        /* is the source below the destination? */

            for( i = count - 1; i >= 0; i--)    /* yes, move from the ends */
               dest[ i] = src[ i];                          /* toward the beginnings */
   }
   else
   {                /* no, the source is above the destination */
            for( i = 0; i < count; i++)         /* move from the beginning toward */
               dest[ i] = src[ i];              /* the end */
   }
}

// END MOVESTR.C
```

```
/*----- analdb() ----------------------------------------------*/
/* Read the file containing the database of routers and variables to
monitor and create router descriptor blocks for use by the programs.
Return NULL on error, or a pointer to the pointers to the blocks.

*/

#include   <malloc.h>              /* for 'malloc()' */
#include   <stdio.h>  /* for 'fprintf()' and 'fileno()' */

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for routinfo struct */


struct routinfo *analdb( dbfilename)
    char *dbfilename;
{
  extern int univerr;

  void *malloc( size_t);
  long filelength( int);

  void free( void *);

  struct routinfo *fillcmunblk( char **);
  FILE *fd;
  char *filebuf, *endatap, *bytep;
  long flen;
  int buflen, errorflag;
  struct routinfo *firstblock, *curblock, *nxtblock;

  errorflag = univerr = 0;
  firstblock = NULL;

  if( ( fd = fopen( dbfilename, "r")) == NULL)
  {       /* file exists? */
          fprintf( stderr, "analdb: can't open file %s\n", dbfilename);
          univerr = -1;
          return( NULL);                          /* no, error */
  }

  if( ( flen = filelength( fileno( fd))) < 0L)
  {       /* got file length? */

          fprintf( stderr, "analdb: file length error is %ld\n", flen);
          fclose( fd);
          univerr = -2;
          return( NULL);                                  /* no, error */
  }

  if( flen > 32767L)
  {       /* file too large for fread()? */
          fprintf( stderr, "analdb: database file is too large, flen = %ld\n",flen);
          fclose( fd);
          univerr = -3;
          return( NULL);
  }

  flen++;             /* add room for a '\0' */

  if( ( filebuf = malloc( ( unsigned int) flen)) == NULL)
  { /* got memory? */
          fprintf( stderr, "analdb: malloc failure\n");
          fclose( fd);
          univerr = -4;
          return( NULL);
  }

  if( ( buflen = fread( filebuf, ( unsigned int) sizeof( char),
                                                  ( unsigned int) flen, fd)) < 0)
  {
          fprintf( stderr, "analdb: file read error\n");
```

```
            fclose( fd);
            free( filebuf);

            univerr = -5;
            return( NULL);
    }

                                    /* parse the configuration file */
    bytep = filebuf;                /* start at the file's beginning */
    endatap = filebuf + buflen;     /* mark the file's end */
    *endatap = '\0';                /* null terminate the ascii data */

    while( bytep < endatap)
    {       /* while there is data in the buffer, loop */
            if( ( nxtblock = fillcmunblk( &bytep)) == NULL)
            {

                    errorflag = -6;
            }
            else
            {
                    if( univerr != 0)               /* any non-fatal error? */
                        errorflag = -7;             /* yes, save it */

                    if( firstblock == NULL)
                    {       /* are there any blocks saved? */
                            firstblock = nxtblock;          /* no, make this the first */

                    }
                    else
                    {       /* there are other blocks in the chain already */
                            curblock->nxtblock = nxtblock;  /* add this one to the chain */

                    }

                    while( nxtblock->nxtblock != NULL)
                    { /* find the end of the */

                            nxtblock = nxtblock->nxtblock; /* chain of added blks */
                    }

                    curblock = nxtblock;                    /* point to the new block */
                    curblock->nxtblock = NULL;              /* initialize its pointer */


            }
    }
    fclose( fd);

    free( filebuf);

    if( errorflag != 0)
            univerr = errorflag;
    return( firstblock);
}

// END ANALDB.C
```

```
/*---- encdindex() ----------------------------------------*/
/* Given a pointer to a null terminated ascii string of characters,
convert the string into an ASN.1 encoded index for an object identifier
and put it int the buffer indicated. Return the number of octets in
the encoded index, or ERROR on error.

*/

#include   <stdio.h>                                   /* for 'fprintf()', NULL */

#include   "\gary\snmp\snmp_src\snmp\snmp.h"           /* for STRINGSIZ */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for objinfo struct */

#define    ERROR              ( -1)                   /* error returned status */

int encdindex( srcp, dstp, dstlen, structp)
  char *srcp;
  unsigned char *dstp;
  int dstlen;
  struct objinfo *structp;
{
  int encdsubid( unsigned long, unsigned char *, int);
  int atoi( const char *);
  unsigned int atohex( char *);

  struct objinfo *objinfop;
  int numcnt, numflag, i, tempint /*, objnamelen */;
  char tempbuf1[ STRINGSIZ];
  unsigned char *nxtencdp, tempbuf2[ STRINGSIZ], *bytep;
  objinfop = structp;

  while( objinfop->flag >= 0)
  {       /* find the array backward pointer so we know what type this is */

          objinfop--;
  }

  for( numflag = numcnt = i = 0 ; ( srcp[ i] != '\0') && ( i < dstlen) ; i++)
  {
        if( ( srcp[ i] == '\t') || ( srcp[ i] == ' ') || ( srcp[ i] == '\056') || ( srcp[ i] == ':') || ( srcp[ i] == '-') || ( srcp[ i] == ','))
        {           /* is it a separator character? */

                    if( numflag != 0)
                    {                   /* yes, have we started a number? */
                              numflag = 0;                    /* yes, clear the number flag */
                              numcnt++;           /* increment the number cnt */
                              tempbuf1[ i] = '\0';      /* terminate the ascii string */
                    }                             /* do nothing if no number started */
        }
        else
        {           /* it is an ascii number character */

                    tempbuf1[ i] = srcp[ i];                  /* copy the character */
                    numflag++;                                /* set the number flag */
        }
  }

  if( srcp[ i] != '\0')    /* processed the entire input string? */
          return( ERROR - 1);                       /* no, error */

  tempbuf1[ i] = '\0';              /* terminate the last string */
  numcnt++;                                 /* add the last number into the count */

  if( ( numcnt < objinfop->type) && ( objinfop->type == 8))
  {
                                            /* is it an atEntry variable? */
          unsigned char *destp,             /* yes, adjust it */
                        *sourcep,
                        *stopptr;

          if( i >= ( STRINGSIZ - 2))      /* enough space to add the separator? */
```

```
            return( ERROR - 3);          /* no, error */

                /* move the strings after the first up two bytes */

          for( stopptr = tempbuf1; *stopptr != '\0'; stopptr++);
            destp = ( unsigned char *) &( tempbuf1[ i + 2]);

          sourcep = ( unsigned char *) &( tempbuf1[ i]);

          while( sourcep > stopptr)
              *( destp--) = *( sourcep--);

          *( destp--) = '\0';     /* insert the string terminator */
          *destp = '1';                   /* insert the separator */
          numcnt++;                       /* increment the number of integers count */
}

if( numcnt != objinfop->type)
{       /* are there the right number of characters in the index for this variable type? */

          fprintf( stderr, "encindex: index character count, %d, is not %d\n",  numcnt, objinfop->type);

          fprintf( stderr, "\tvariable is %s\n", structp->objname);

          return( ERROR - 2);
}

nxtencdp = tempbuf2;              /* point at the second temporary buffer */
numflag = numcnt;                 /* get the count of integers/strings */

for( i = 0 ; numflag > 0 ; i++, numflag--)
{       /* convert ascii numbers to integers and encode them according to ASN.1 */
        int j;

                          /* find any hexadecimal characters in the string */

        for( j = 0; ( tempbuf1[ i + j] >= '0') && ( tempbuf1[ i + j] <= '9')
                        && ( tempbuf1[ i + j] != '\0'); j++);

        if( tempbuf1[ i + j] != '\0')
        {           /* are there are hex characters? */

          /* convert string to hex? */
            tempint = atohex( tempbuf1 + i);          /* yes, convert the hex string
                                                          to an integer */
        }
        else
        {                 /* there are only integer characters */
            tempint = atoi( tempbuf1 + i);            /* convert string to an int */
        }

          while( tempbuf1[ i] != '\0')      /* find the end of the string */
              i++;

                          /* encode in ASN.1 notation the resultant integer */


        if( ( tempint = encdsubid( ( unsigned long) tempint, nxtencdp, ( int)( ( tempbuf2 + STRINGSIZ) - nxtencdp))) < 0)
        {
                                                      /* encoded okay? */
                fprintf( stderr, "encdindex: encdsubid returned %d\n", tempint);
                return( ERROR - 4);               /* no, error */
        }
        else                              /* encoded okay! tempint is the number of
                                             bytes used in the encoded number */

          nxtencdp += tempint;            /* point to the next buffer location
                                             where encodings can be put */
}
if( nxtencdp - tempbuf2 > dstlen)
{       /* enough space for the index? */
```

```
            return( ERROR - 5);                     /* no, error */
    }
                                /* move the encoded bytes to the user's buffer */
    for( i = 0 ; ( ( unsigned char *) &( tempbuf2[ i]) < nxtencdp) && ( i < dstlen) ; i++)
    {
            dstp[ i] = tempbuf2[ i];
    }

    if( ( unsigned char *) &( tempbuf2[ i]) != nxtencdp)    /* moved okay? */
            return( ERROR - 13);                    /* no, error */

    return( i);                     /* return the number of octets in the encoding */
}

// END ENCINDEX.C
```

```
/*----- fillcmunblk() ----------------------------------------*/
/* Given a pointer to a pointer to an ASCII string of router information,
get memory and fill in the various blocks with the appropriate data.
Return NULL on error; return a pointer to the filled-in block on success.
Modify the pointer to the data to point past the used ASCII data on
success.

*/

#include   <stdio.h>                              /* for 'fprintf()', NULL */

#include   "\gary\snmp\snmp_src\snmp\snmp.h"      /* for routinfo struct, STRINGSIZ */

struct routinfo *fillcmunblk( datap)
    char **datap;
{
  extern int univerr;

  int movenstr( unsigned char *, unsigned char *);
  struct routinfo *fillroutblk( char **, char *);
  char *bytep, *strp, *endatap, c, community[STRINGSIZ];

  struct routinfo *firstroutbp,*nxtblock, *curblock;
  int strlen,endflag,errorflag, blankflag;

  univerr = 0;
  firstroutbp = NULL;
  endatap = *datap;

                              /* find the end of the data for this router */

  while( ( *endatap != '\n') && ( *endatap != '\0'))      /* look for '\n' or end of data */
          endatap++;

  if( *endatap == '\0')
  {       /* found end of data before '\n'? */
          *datap = ++endatap;                     /* yes, error */
          univerr = -1;
          return( NULL);                          /* fatal error */
  }

  errorflag = endflag = 0;

  do
  {               /* find end of community data */
    do
    {             /* look for blank ('\t', ' ') or empty lines */
            char *lineend;

            while( ( *endatap != '\n') && ( *endatap != '\0'))          /* find a '\n' */
                    endatap++;

            lineend = endatap++;                                        /* mark the location of the '\n' */
            blankflag = 1;                                              /* note that we found one */

            while( ( *endatap == ' ') || ( *endatap == '\t'))
                    endatap++;                                          /* move past any following spaces or tabs */

            if( ( *endatap != '\n') && ( *endatap != '\0'))             /* found '\n' or end of data? */
                    blankflag = 0;                                      /* no, look for another '\n' */
            else
                    if( *endatap == '\n')                               /* was a '\n' found? */
                      endatap = lineend;                                /* yes, use the first '\n' found */
                        /* else *endatap == '\0', use current endatap */

    } while( blankflag != 1);     /* loop until a blank or empty line is found. endatap now points to the end of data or the first
                                     of two '\n's which may be separated by nothing, spaces, or tabs only */

                                  /* now look for new community data */
                                  /* move past any more blank lines, comment only lines, and whitespace */
```

```
        while( ( ( *endatap == ' ') || ( *endatap == '\n') || ( *endatap == '\043') || ( *endatap == '\t')) && ( *endatap != '\0'))
        {
                if( *endatap == '\043')
                {
                        while( ( *endatap != '\n') && ( *endatap != '\0'))
                                endatap++;
                }
                else
                        endatap++;
        }

        if( *endatap == '\0')   /* are we at the end of data? */
             endflag = 1;       /* yes, we're done */

        else
            if( *( endatap - 1) == '\n')      /* are we at a the beginning of a line which doesn't begin with whitespace? */
                    endflag = 1;                      /* yes, we're done */
            else                              /* not at end of data, not a line with no beginning whitespace */
                    endflag = 0;                      /* continue looping */
} while( endflag != 1);

bytep = *datap;                 /* point to the beginning of data */
                                /* now move past initial white space and comments */

while( ( bytep < endatap) && ( ( *bytep == ' ') || ( *bytep == '\t') || ( *bytep == '\043') || ( *bytep == '\n')))
{
        if( *bytep == '\043')
        {                       /* found a '#'? */
                while( ( *bytep != '\n') && ( bytep < endatap))
                                /* yes, look for an '\n' */
                    bytep++;
        }
        else
            bytep++;
}

if( bytep >= endatap)
{       /* went past end of buffer? */
           *datap = ++endatap;          /* mark the end of processed data */
           univerr = -2;                /* yes, found no router address, error */
           return( NULL);               /* fatal error */
}

strp = bytep;                   /* mark beginning of the community string */

if( *strp == 0x22)
{       /* found a ""? */
           strp++;                      /* point past it */
           bytep++;
                                        /* look for another "" */
           while( ( bytep < endatap) && ( *bytep != 0x22))
                bytep++;
}
else
{
                                                /* move to first white space */
           while( ( bytep < endatap) && ( *bytep != ' ') && ( *bytep != '\t') && ( *bytep != '\n'))
                bytep++;
}

if( ( bytep - strp) >= STRINGSIZ)
{       /* community string too large? */
           univerr = -3;                /* yes, error */
           return( NULL);               /* fatal error */
}

c = *bytep;
*bytep = '\0';

 movenstr( ( unsigned char *) strp, ( unsigned char *) &( community[0]));
 *bytep = c;
```

```
    if( c == 0x22)
            bytep++;

    while( bytep < endatap)
    {       /* while there is data in the buffer, loop */
            if( ( nxtblock = fillroutblk( &bytep, endatap)) == NULL)
            {
                    errorflag = -6;
            }
            else
            {
                    if( univerr != 0)
                    {                                       /* any non-fatal error? */
                            errorflag = -7;                 /* yes, save it and continue */
                    }

                    if( firstroutbp == NULL)
                    {       /* first block filled? */
                          firstroutbp = nxtblock;           /* yes, this is the first */
                    }
                    else
                    {                                       /* not the first block to be filled */
                            curblock->nxtblock = nxtblock;  /* add it to the chain */
                    }
                    curblock = nxtblock;                    /* point to the new block */
                    curblock->nxtblock = NULL;              /* terminate the chain at the new blk */
                                        /* move the community string into the new block */
                    movenstr( ( unsigned char *) &( community[0]), ( unsigned char *) &( curblock->community[0]));
            }
    }/* while there is data in the buffer, loop */

    if( *endatap != '\0')
    {       /* point past the data just analyzed */

            while( *endatap == '\n')
                endatap++;
    }
    *datap = endatap;               /* mark the end of processed data */

    if( errorflag != 0)                     /* any non-fatal errors reported? */
            univerr = errorflag;            /* yes, pass the last one on */

    return( firstroutbp);                   /* return the head of the chain of blocks */
}

// END FLCMUNBL.C
```

```
/*----- fillroutblk() ----------------------------------------------*/
/* Given a pointer to a pointer to an ASCII string of router information,
get memory and fill in the various blocks with the appropriate data.
Return NULL on error; return a pointer to the filled-in block on success.
Modify the pointer to the data to point past the used ASCII data on
success.

*/

#include   <malloc.h>                                   /* for 'malloc()' */
#include   <stdio.h>  /* for 'fprintf()', NULL */

#include   "\gary\snmp\snmp_src\snmp\snmp.h"            /* for routinfo struct */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for objinfo struct */

#include   <stdlib.h>  /* for 'free()' and 'atoi()' */
#include   <io.h>                                       /* for 'filelength()' */

#define    ERROR              (-1)                      /* error returned status */

struct routinfo *fillroutblk( datap, endata)
  char **datap, *endata;
{
  extern struct objinfo *isoorgroot;
  extern int univerr;

  void *malloc( size_t);

  int movenstr( unsigned char *, unsigned char *);
  int encdindex( char *, unsigned char *, int, struct objinfo *);
  int freevblk( struct routinfo *, struct varinfo *);
  int freerblk( struct routinfo *);
  struct objinfo *searcharr( struct objinfo *, char *);

  char *bytep, *strp, *endatap,c;
  struct objinfo *structp;
  struct routinfo *rblkp;
  struct varinfo *vblkp, *nxtvblkp;
  int blankflag;

  univerr = 0;
  endatap = *datap;
  while( ( *endatap == '\n') && ( endatap < endata))          /* skip leading '\n's */
          endatap++;

  if( endatap == endata)
  {       /* got to end of data before the first '\n'? */
          *datap = endata;                                    /* yes, error */
          univerr = -1;
          return( NULL);                                      /* fatal error */
  }

  bytep = endatap;                                            /* start past the leading '\n's */

  do
  {                 /* look for blank ('\t', ' ') or empty lines */
          char *lineend;

          while( ( *endatap != '\n') && ( endatap < endata))  /* find a '\n' */
              endatap++;

          lineend = endatap++;                                /* mark the location of the '\n' */
          blankflag = 1;                                      /* note that we found one */

          while( ( ( *endatap == ' ') || ( *endatap == '\t')) && ( endatap < endata))
              endatap++;                                      /* move past any following spaces or tabs */


          if( ( *endatap != '\n') && ( endatap < endata))     /* found '\n' and not end of data? */
            blankflag = 0;                                    /* no, look for another '\n' */
          else
```

```
            if( *endatap == '\n')                       /* was a '\n' found? */
                endatap = lineend;                      /* yes, use the first '\n' found as end of data */
                        /* else *endatap == endata, use current endatap */

} while( blankflag != 1);                               /* loop until a blank or empty line is found */

        /* endatap now points to the end of data or the first of two '\n's which may be separated by nothing, spaces, or tabs only */
                        /* move past initial white space and comments */

while( ( bytep < endatap) && ( ( *bytep == ' ') || ( *bytep == '\t') || ( *bytep == '\043') || ( *bytep == '\n')))
{
        if( *bytep == '\043')
        {                       /* found a '#'? */
                while( ( *bytep != '\n') && ( bytep < endatap))
                                                /* yes, look for an '\n' */

                        bytep++;
        }
        else
            bytep++;
}

if( bytep >= endatap)
{       /* went past end of buffer? */
          *datap = ++endatap;           /* skip the rest of this router's data */
          univerr = -2;                 /* yes, found no router address, error */
          return( NULL);                /* fatal error */
}

strp = bytep;                   /* mark beginning of the address string */

if( *strp == 0x22)
{
          strp++;
          bytep++;
          while( ( bytep < endatap) && ( *bytep != 0x22))
              bytep++;
}
else
{
                        /* found first char of router addr */
                /* move past characters to first white space */

          while( ( bytep < endatap) && ( *bytep != ' ') && ( *bytep != '\t') && ( *bytep != '\n'))
              bytep++;
}

if( bytep >= endatap)
{               /* went past end of buffer? */

          *datap = ++endatap;           /* yes, skip the rest of this router's data */
          univerr = -5;                         /* error */
          return( NULL);                /* fatal error */
}
                /* found end of router address */
if( bytep - strp > 15)
{       /* address string too long? */
          *datap = ++endatap;           /* skip the rest of this router's data */
          univerr = -6;                 /* yes, error */
          return( NULL);                /* fatal error */
}
                                        /* get memory for a router address block */
if( ( rblkp = ( struct routinfo *) malloc( sizeof( struct routinfo))) == NULL)
{
          univerr = -7;                 /* failed to get memory? */
          return( NULL);                /* yes, fatal error */
}                                       /* initialize address block */


rblkp->nxtblock = NULL;                 /* initialize address block pointers */
rblkp->vblockp = NULL;
```

```
c = *bytep;                                     /* save char after router address */

*bytep = '\0';                          /* mark end of router address string */

movenstr( ( unsigned char *) strp, ( unsigned char *) &( rblkp->routeraddr[0]));
                                        /* move the address string into the address block */

*( bytep++) = c;                        /* restore overwritten character */

while( bytep < endatap)
{                       /* loop while there is variable data */
        while ( ( bytep < endatap) && ( ( *bytep == ' ') || ( *bytep == '\t') || ( *bytep == '\043') || ( *bytep == '\n')))
        {               /* move past comments and white space */

                if( *bytep == '\043')
                {                               /* found a '#'? */
                        while( ( *bytep != '\n') && ( bytep < endatap))
                                                /* yes, look for an '\n' */
                                bytep++;
                }
                else
                    bytep++;
        }
        if( bytep >= endatap)
        {                               /* went past end of buffer? */
                if( rblkp->vblockp == NULL)
                {               /* yes, no variables found? */

                        freerblk( rblkp);                       /* free the unused blk */

                        rblkp = NULL;
                        univerr = -8;                           /* set an error flag */
                        *datap = ++endatap;     /* skip the router's data */
                }               /* else, variables found, leave them as processed */

                break;                                  /* done, get out of 'while' loop */
        }                       /* now we are past the white space, and hope to be pointing at a variable name */
        strp = bytep;                           /* mark the beginning of the string */

        if( *strp == 0x22)
        {
             strp++;
             bytep++;
             while( ( bytep < endatap) && ( *bytep != 0x22))
                        bytep++;
        }
        else
        {
                while( ( bytep < endatap) && ( *bytep != ' ') && ( *bytep != '\t') && ( *bytep != '\n'))
                                                                                        /* move to first white space */
                  bytep++;
        }
        if( bytep >= endatap)                           /* went past end of buffer? */
                bytep = endatap;                        /* point to end of variable name */

        if( bytep - strp >= STRINGSIZ)
        {               /* string too long? */
            univerr = -10;                      /* set a non-fatal error flag */
        }
        else
        {
                c = *bytep;
                *bytep = '\0';

                if( rblkp->vblockp == NULL)
                {               /* is this the first variable block? */

                        if( ( rblkp->vblockp =  (struct varinfo *) malloc( sizeof( struct varinfo)))== NULL)
                        {
                                /* yes, got a variable block of memory? */
```

```
                        freerblk( rblkp);       /* free the unused blk */

                        rblkp = NULL;
                        univerr = -11;                  /* fatal error */
                        break;
                }
                vblkp = rblkp->vblockp;         /* got blk, point to it */
}
else
{                       /* not the first variable block */
                if( ( vblkp->nxtblock = (struct varinfo *) malloc( sizeof( struct varinfo))) == NULL)
                {
                                /* got variable block memory? */
                        univerr = -12;                  /* no, fatal error */
                        return( rblkp);
                }
                vblkp = vblkp->nxtblock;        /* got blk, point to it */
}                       /* initialize this variable block */
vblkp->nxtblock = NULL;
vblkp->vartype = 0;
vblkp->indexlen = 0;
vblkp->changeflag = 0;                          /* clear change flag */
vblkp->timestamp = 0L;                          /* clear time stamp */
vblkp->varvalue = NULL;
structp = isoorgroot;

if( ( structp = searcharr( structp, strp)) == NULL)
{
                univerr = -13;          /* is the variable part of the MIB? */
                movenstr( ( unsigned char *) strp, ( unsigned char *) &( vblkp->variable[0]));
                                /* no, move the string to the blk */

                *(bytep++) = c;         /* go to next variable on the next line */

                if( c != '\n')
                {               /* are we at the end of the line? */

                        while( ( *bytep != '\n') && ( bytep < endatap))             /* no */
                                bytep++;        /* find it */
                }
}
else
{                       /* variable string is part of the MIB */
                int varstorage;
                switch( ( unsigned char) structp->type)
                {
                        case( UINTID):
                                                varstorage = VTYPEINT;
                                                break;
                        case( UOCTSTRID):
                        case( IntegerStrID):
                                                varstorage = VTYPESTR;
                                                break;
                        case( DisplStrID):
                                                varstorage = VTYPEDSTR;
                                                break;
                        case( IpAddrID):
                                                varstorage = VTYPEIPAD;
                                                break;
                        case( CounterID):
                                                varstorage = VTYPECTR;
                                                break;
                        case( TimeTicksID):
                                                varstorage = VTYPETIME;
                                                break;
                        case( GaugeID):
                                                varstorage = VTYPEGAGE;
                                                break;
                        case( UOBJIDID):
                                                varstorage = VTYPESTR;
                                                break;
```

```
            default:
                                varstorage = 0;
}
vblkp->vartype = structp->type;   /* set variable type */

if( varstorage == 0)
            vblkp->varvalue = NULL;
else
{
            if( ( vblkp->varvalue = malloc( varstorage)) == NULL)
            {
                        fprintf( stderr, "fillroutblk: variable storage malloc error\n");
                        freevblk( rblkp, vblkp);
                        univerr = -13;
            }
}
if( ( varstorage == VTYPESTR) || ( varstorage == VTYPEDSTR))
            *( ( int *) vblkp->varvalue) = 0;
movenstr( ( unsigned char *) strp,

( unsigned char *) &( vblkp->variable[0]));

                        /* move the string to the blk */

*( bytep++) = c;        /* restore the overwritten char */

if( ( c != '\n') && ( c != '\0'))
{           /* line ends?
            /* skip the white space before the index */

            while( ( bytep < endatap) && ( ( *bytep == ' ') || ( *bytep == '\t')))
                                                            /* move beyond white space */
                        bytep++;
            if( *bytep == '\043')
            { /* found comment? (no index value?) */

                        while( ( *bytep != '\n') && ( bytep < endatap))
                                    bytep++;
            }
            else
            {
                                                /* there is an index value, get it */
                        strp = bytep;           /* mark the beg of the index string */
                        if( *strp == 0x22)
                        {
                                    strp++;
                                    bytep++;
                                    while( ( bytep < endatap) && ( *bytep != 0x22))
                                                bytep++;
                        }
                        else
                        {
                                    while( ( bytep < endatap) && ( *bytep != ' ') && ( *bytep != '\t')
                                                                    && ( *bytep != '\n'))

                                    bytep++;
                        }
                        if( bytep >= endatap)               /* string terminated by end-of-data? */
                                    bytep = endatap;        /* yes, mark the end of string */

                        /* assume ascii string will be longer than the encoded string */

                        if( bytep - strp >= INDEXSIZ)
                        {
                                    /* index str too long? */
                                    freevblk( rblkp, vblkp);

                                    if( ( vblkp = rblkp->vblockp) != NULL)
                                    {
                                                while( vblkp->nxtblock != NULL)
                                                            vblkp = vblkp->nxtblock;
                                    }
```

```
                                        univerr = -14;          /* non-fatal error */
                                }
                                else
                                {               /* index string will fit */
                                        int status;
                                        c = *bytep;             /* save string ending char */
                                        *bytep = '\0';          /* mark the end of the index string */
                                        if( ( status = encdindex( strp, ( unsigned char *) &
                                                ( vblkp->index[0]), ( int) INDEXSIZ, structp)) < 0)
                                        {
                                        /* string converted to ASN.1 index? */
                                             fprintf( stderr, "fillroutblk: encdindex error status was %d\n",
                                                                                        status);
                                              freevblk( rblkp, vblkp);
                                              if( ( vblkp = rblkp->vblockp) != NULL)
                                              {
                                              /* is this the first variable block for this router? */

                                                   while( vblkp->nxtblock != NULL)
                                                   vblkp = vblkp->nxtblock;
                                               }
                                               univerr = -15;  /* non-fatal error */
                                        }
                                        else
                                        {       /* string was converted to ASN.1 */
                                                vblkp->indexlen = ( unsigned char) status;
                                                /* save the number of octets in result */
                                        }
                                        *bytep = c;             /* restore the overwritten character */
                                        bytep++;  /* point past the end of the
                                                        index string */
                                }
                        }
                }
        }
  }
  if( rblkp->vblockp == NULL)
  {
        freerblk( rblkp);
        rblkp == NULL;
  }

  if( *endatap == '\n')
        endatap++;
  *datap = endatap;
  return( rblkp);
}

// END FLROUTBL.C
```

```
/*----- freerblk() ----------------------------------------*/
/* Given a pointer to a router info block, free the block, fix the
chain of router info block pointers, and free any variable info
blocks associated with the router info block.  Return 0 on success;
ERROR on error.

*/

#include    <stdio.h>                                   /* for 'fprintf()', NULL */

#include    "\gary\snmp\snmp_src\snmp\snmp.h"           /* for routinfo struct */

int freerblk( routblkp)
   struct routinfo *routblkp;
{
   extern struct routinfo *firstrinfop;
   void free( void *);
   struct routinfo *rblkp;
   struct varinfo *vblkp;
   rblkp = firstrinfop;

   while( ( rblkp->nxtblock != routblkp) && ( rblkp->nxtblock != NULL))
   {
                                /* is this the router info block to be released? */
            rblkp = rblkp->nxtblock;                   /* no, point to the next one */
   }

   if( rblkp != routblkp)                              /* found the right block? */
            return( 0);                                /* no, forget it */

   if( rblkp == firstrinfop)                           /* is the right block the first one? */
            firstrinfop = routblkp->nxtblock;          /* yes, adjust the pointers */
   else                                                /* the right block is in the chain somewhere */
            rblkp->nxtblock = routblkp->nxtblock;      /* remove this block */

   while( ( vblkp = routblkp->vblockp) != NULL)        /* free any variable info blocks associated with this router info block */
            freevblk( routblkp, vblkp);

   free( routblkp);                    /* free the router info block */

   return( 0);
}

// END FREERBLK.C
```

```
/*----- freevblk() ----------------------------------------*/
/* Given a pointer to a router info block and one of its variable
info blocks that is to be freed, free the variable info block and
fix the chain of variable info block pointers associated with the
router info block.  Return 0 on success; ERROR on error.

*/

#include   <stdio.h>                                  /* for 'fprintf()', NULL */

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for routinfo struct */

int freevblk( routblkp, varblkp)
   struct routinfo *routblkp;
   struct varinfo *varblkp;
{
   void free( void *);
   struct varinfo *vblkp;

   if( routblkp->vblockp == varblkp)                  /* is this the first variable block for this router? */
            routblkp->vblockp = varblkp->nxtblock;    /* yes, release it */
   else
   {                                  /* not the first var blk, find it */
            vblkp = routblkp->vblockp;                /* start at the top of the chain */

            while( ( vblkp->nxtblock != varblkp) && ( vblkp->nxtblock != NULL))
                     vblkp = vblkp->nxtblock;

            if( vblkp->nxtblock != varblkp)           /* found the right block? */
              return( 0);                                     /* no, forget it */

            vblkp->nxtblock = varblkp->nxtblock;              /* remove the block from the chain of blocks */
   }
   free( varblkp);                                            /* free the block */

   return( 0);
}

// END FREEVBLK.C
```

```
/*----- atohex() ----------------------------------------------------*/
/* Given a pointer to an ascii string of hexadecimal digits, convert
them to an integer of the correct value.  No error retur.

*/

#define     ERROR               (-1)        /* error returned status */
#define     UCASEMSK            0x5f        /* mask converting char to upper case */

unsigned int atohex( strp)
      char *strp;
{
   int i, j;
   unsigned int value;

   for( j = 0 ; strp[j] != '\0' ; j++);                    /* find the end of the supplied string */

   if( j > sizeof( value))                                 /* is it too big? */
            return( ERROR);                                /* yes, return -1 */

   j--;                                                    /* point at the last character */

   if( j < 0)                                              /* are there no characters? */
           return( ERROR);                                 /* right, return -1 */

   value = 0;

   for( i = 0 ; j >= 0 ; i++, j--)
   {        /* process the chars */
            if( ( strp[j] >= '0') && ( strp[j] <= '9'))             /* is the char 0-9? */
               value += ( int) ( strp[j] - '0') * ( 1 << ( i * 4));
                                                                    /* yes, evaluate it */
            else
            {                   /* the character isn't 0-9 */

               strp[j] &= UCASEMSK;                                 /* make it upper case */

               if( ( strp[j] >= 'A') && ( strp[j] <= 'F'))          /* is it A-F? */
                        value += ( ( int) ( strp[j] - 'A') + 10) * ( 1 << ( i * 4));
               else             /* it isn't 0-9 or A-F */

                        return( value);                 /* return what we have so far */
            }
   }
   return( value);
}

// END ATOHEX.C
```

```
/*---- checkcmun() ----------------------------------------*/
/* Check the community octet pointed to by 'cmuntagp' for a match
with the community of the network descriptor designated by nd.
Return NULL on mismatch, or a pointer to the pdu tag on a match.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */
#include   <stdio.h>                                  /* for 'fprintf()' and NULL */

unsigned char *checkcmun( cmuntagp, nd)
      unsigned char *cmuntagp;
      int nd;
{
  int cmpstr( unsigned char *, unsigned char *, int);
  int getobjptrs( unsigned char *, struct objptrs *);
  unsigned char *getnxtobj( unsigned char *);
  unsigned char *getcmunp( int);
  long getconnreqid( int);
  int len;
  unsigned char *datap, *cmunp;
  struct objptrs objp;

  if( *cmuntagp != UOCTSTRID)              /* correct community tag? */
            return( NULL);                 /* no, error */

  if( getobjptrs( cmuntagp, &objp) < 0)    /* got object pointers? */
            return( NULL);                 /* no, error */

                  /* don't check the community if this is a trap packet */

  if( getconnreqid( nd) > ( long)( -2))
  {       /* is this not a trap pkt? */

            datap = objp.datap;                        /* right, check the community */
            len = ( int)( objp.endatap - datap);

            if( ( cmunp = getcmunp( nd)) == NULL)      /* got community pointer? */
              return( NULL);                           /* no, error */

            if( cmpstr( datap, cmunp, len) != 0)       /* community names match? */
              return( NULL);                           /* no, error */
  }

  if( ( datap = getnxtobj( objp.tagp)) == NULL)        /* got next object? */
            return( NULL);                             /* no, error */

  return( datap);       /* return a pointer to the next obj tag */
}

// END CHECKCMU.C
```

```
/*---- checkpkt() ----------------------------------------*/
/* Check the packet in 'buffer' of length 'bytes' for valid SNMP
format. The Ethernet, IP, and UDP headers are not in 'buffer'. The
first byte of 'buffer' is the first data byte. Return NULL on
error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */
#include   <stdio.h>                                  /* for 'fprintf()' and NULL */

unsigned char *checkpkt( buffer, bytes, nd)
      unsigned char *buffer;
      int bytes, nd;
{
  int getobjptrs( unsigned char *, struct objptrs *);
  unsigned char *checkver( unsigned char *);
  unsigned char *checkcmun( unsigned char *, int);
  unsigned char *bytep,*versionp,*communityp,*pdup;

  struct objptrs msghdr;
  int len;

  if( getobjptrs( buffer, &msghdr) < 0)
  {         /* got object pointers? */

            return( NULL);                           /* no, error */
  }
  if( *msghdr.tagp != USEQID)
  {         /* is this a proper message? */
            return( NULL);                           /* no, error */
  }

  if( ( int)( msghdr.endobjp - msghdr.tagp) != bytes)
  {
                                                     /* buffer data length = message length? */
            return( NULL);                           /* no, error */
  }

  versionp = msghdr.datap;                           /* point to the version tag byte */

  if( ( communityp = checkver( versionp)) == NULL)
                                                     /* correct version of SNMP? */
            return( NULL);                           /* no, error */

  if( ( pdup = checkcmun( communityp, nd)) == NULL)
                                                     /* correct SNMP community? */
            return( NULL);                           /* no, error */

  return( pdup);                                     /* return a pointer to the pdu */
}

// END CHECKPKT.C
```

```
/*---- checkver() ----------------------------------------*/
/* Check the version field pointed to by the pointer of this
message against the version that this software is compatible with.
Return a pointer to the community tag. Return NULL on error.

*/

#include     "\gary\snmp\snmp_src\snmp\snmp.h"
#include     <stdio.h>                    /* for 'fprintf()' and NULL */

unsigned char *checkver( verp)
     unsigned char *verp;
{
  int getobjptrs( unsigned char *, struct objptrs *);
  unsigned char *getnxtobj( unsigned char *);
  struct objptrs objp;

  if( *verp != UINTID)                    /* correct tag for version object? */
          return( NULL);                  /* no, error */

  if( getobjptrs( verp, &objp) < 0)       /* got object pointers? */
          return( NULL);                  /* no, error */

  verp = objp.datap;                      /* point to the data */

  if( *verp != SNMPVER)                   /* correct SNMP version? */
          return( NULL);                  /* no, error */

  if( ( verp = getnxtobj( objp.tagp)) == NULL)          /* got next object? */
          return( NULL);                                /* no, error */

  return( verp);                          /* point to the community tag */
}
// END CHECKVER.C

/*---- decdsubid() ---------------------------------------*/
/* Given a pointer to a pointer to an ASN.1 encoded sub-identifier
in an object identifier, decode it. Adjust the pointer to the
encoding so it points to the first octet past the encoding. Return
ERROR on error; the decoded long integer on success.
*/

#include   <stdio.h>                      /* for fprintf() */

long decdsubid( subidp)
   unsigned char **subidp;
{
  unsigned char *bytep;
  int i, j;
  long value;
  bytep = *subidp;                        /* point to the encoding */

  if( ( *bytep & 0x80) == 0)
  {       /* is the encoding a single byte? */
          (*subidp)++;                    /* yes, adjust the pointer */
          return( ( long) *bytep);
  }
                  /* multiple byte encoding */
  value = 0L;                             /* clear the returned value */

  for( i = 0; ( *bytep & 0x80) != 0 ; i++, bytep++);      /* find the end of  this encoding */
          *subidp = bytep + 1;                            /* save this location */

  for( j = 0 ; i >= 0 ; i--, j++, bytep--)                /* put the bits into 'value' in their correct order */
          value = value | ( ( ( long) ( *bytep & 0x7f)) << ( 7 * j));
  return( value);
}
// END DECSCUBID.C

/*---- encdsubid() ---------------------------------------*/
/* Given a positive integer, encode it to become an object identifier
```

```
sub-identifier according to ASN.1 encoding rules. But the encoded bytes
in the designated buffer. Return the number of bytes in the encoding
on success; ERROR on error.

*/

#include   <stdio.h>  /* for fprintf() */

#define    ERROR              ( -1)

int encdsubid( value, buffer, bufsize)
  unsigned long value;
  unsigned char *buffer;
  int bufsize;
{
  int i,
        shftcnt;

  if( value < 128L)
  {
          buffer[0] = ( unsigned char) value;
          return( 1);
  }
                                        /* encode 'value' into an ASN.1 integer */

  for( i = bufsize - 1, shftcnt = 0 ; ( shftcnt < ( sizeof( value) * 8)) && ( i >= 0) ; i--, shftcnt += 7)
  {
        buffer[i] = value & 0x7f;       /* mask in the lowest 7 bits */

          if( shftcnt != 0)             /* is this the least signif. byte? */
            buffer[i] |= 0x80;          /* no, OR in 0x80 */

          value = value >> 7;           /* shift the value 7 bits right */
  }

  if( i < 0)
          return( ERROR);

  i++;

  while( ( buffer[i] == 0x80) && ( i < bufsize))        /* while the leading byte is 0x80, drop it */
          i++;

  for( shftcnt = 0 ; i < bufsize ; shftcnt++, i++)      /* move the bytes to the beginning of the buffer */
          buffer[shftcnt] = buffer[i];

  return( shftcnt);
}

// END ENCSUBID.C
```

```
/*---- finddatafld() ----------------------------------------*/
/* Given a pointer to a length field, return a pointer to the data
field. Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */

unsigned char *finddatafld( lenp)
     unsigned char *lenp;
{
                                   /* lenp points to the first length field byte */

  if( *lenp == FLDCONMSK)    /* constructed length? */
            lenp++;                        /* yes, the data begins at the next byte */
  else
  {
          if( ( *lenp & FLDCONMSK) == FLDCONMSK)
          {          /* multi-byte field? */

                     int len;                          /* yes */

                     len = *lenp & ~( FLDCONMSK); /* get the number of additional bytes in the length field */

                     lenp += len + 1;       /* point to the first data field byte */
          }
          else                               /* single byte length field */
              lenp++;                        /* point to the first data field byte */
  }
  return( lenp);                             /* return the pointer to the first data byte */
}

// END FNDDATFL.C

/*---- findlenfld() -----------------------------------------*/
/* Given a pointer to a tag, return a pointer to the length field.
Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */

unsigned char *findlenfld( tagp)
     unsigned char *tagp;
{
                                                          /* tagp points to the first tag field byte */

  if( ( *tagp & IDTAGMSK) == IDTAGMSK)
  {        /* multi-byte tag? */

            tagp++;                                       /* yes, point to the first byte of the tag */
            while( ( *tagp & FLDCONMSK) == FLDCONMSK)
            tagp++;                                  /* while the tag continuation bit is true, move to the to the next byte */
            tagp++;                                  /* point to the first length byte */
  }
  else                          /* single byte tag */
         tagp++;                /* point to the first length byte */
  return( tagp);                /* return the pointer to the first len byte */
}

// END FNDLENFL.C
```

```
/*----- getintval() ----------------------------------------*/
/* Get the value of the the integer object encoding pointed to by
the pointer passed to the function.  Put the integer value in the
location also passed to the function.  Return zero on success or
ERROR on error.

*/

#include    "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */

#define     ERROR               ( -1)

int getintval( obj, intaddr, intsize)
   unsigned char *obj;
   void *intaddr;
   int intsize;
{
   int getobjptrs( unsigned char *, struct objptrs *);
   struct objptrs objp;
   unsigned char *bytep;
   int i;
   long value;

   if( ( *obj != UINTID) && ( *obj != CounterID) && ( *obj != GaugeID) && ( *obj != TimeTicksID))
                                        /* pointing at an integer encoding? */
            return( ERROR);             /* no, error */

   if( intsize > sizeof( value))        /* integer too long? */
               return( ERROR - 1);      /* yes, error */

   if( getobjptrs( obj, &objp) < 0)     /* got object pointers? */
               return( ERROR - 1);      /* no, error */

   value = 0L;                          /* clear the variable value */
   bytep = objp.datap;                  /* point to the data */

   for( i = 0 ; ( i < intsize) && ( bytep < objp.endatap) ; i++, bytep++)
            value = ( value << 8) | ( long) *bytep;

   if( bytep != objp.endatap)           /* too much data? */
               return( ERROR - 3);      /* yes, error */

   if( intsize == sizeof( int))
               *( ( int *)intaddr) = ( int) value;
   else
               *( ( long *)intaddr) = value;

   return( 0);                          /* no error, return success */
}

// END GETINTVL.C
```

```
/*---- getnxtobj() ----------------------------------------*/
/* Given a pointer to an object, return a pointer to the next
object.  Return NULL on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */
#include   <stdio.h>  /* for NULL */

unsigned char *getnxtobj( obj)
      unsigned char *obj;
{
   int getobjptrs( unsigned char *, struct objptrs *);
   struct objptrs objp;

   if( getobjptrs( obj, &objp) < 0)              /* got obj pointers? */
            return( NULL);                       /* no, error */

   return( objp.endobjp);                        /* return location of next object */
}

// END GETNXTOB.C
```

```
/*---- getobjlen() ----------------------------------------*/
/* Get the length of the data field in bytes of from the length field
pointed to by length field pointer.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for FLDCONMSK */
#include   <stdio.h>  /* for NULL */

#define    ERROR              ( -1)

int getobjlen( lenp)
  unsigned char *lenp;
{
  unsigned char *finddatafld( unsigned char *);
  unsigned char *bytep;
  int lenlen, i;
  unsigned int len, olen;

                              /* lenp points to the first length field byte */

  if( *lenp == FLDCONMSK)
  {       /* is the length constructed? */

          if( ( bytep = finddatafld( lenp)) == NULL) /* yes, found data fld? */
               return( ERROR);              /* no, error */

          for( i = 0, len = 0 ; i < 2 ; bytep++, len++)
          {          /* look for consecutive nulls for the 'end-of-data' */

                    if( *bytep == '\0')              /* found a null? */
                         i++;                        /* yes, increment i */
                    else
                         i = 0;                      /* otherwise, clear i */

                                                     /* found the 'end-of-data' */
          }

          return( len);                              /* return the length value */
  }

  if( ( *lenp & FLDCONMSK) != FLDCONMSK)
  {       /* multi-byte field? */
             return( ( unsigned int) *lenp);         /* no, return the length */
  }
                              /* multi-byte length field */
                    /* get the number of additional bytes in the length field */


  lenlen = ( int) ( *lenp & (( unsigned char) ~( FLDCONMSK)));
  lenp += lenlen;                                    /* point to the last length field byte */
  olen = len = ( unsigned int) *lenp;                /* get its value */
  lenp--;                                            /* point to the next to last len field byte */
  lenlen--;                                          /* decrement the number of bytes left in the field */

  for( i = 1 ; lenlen > 0 ; i++, lenlen--, lenp-- )
  {       /* do it again */
             len += ( unsigned int)( ( ( unsigned int) *lenp) << ( 8 * i));
             if( len < olen)                         /* partial test for overflow? */
             return( ERROR);                         /* yes, error */
             olen = len;
  }

  return( len);
}

// END GETOBJLN.C

/*---- getobjptrs() ---------------------------------------*/
/* Given a pointer to an object and a structure of pointers
concerning that object, fill in the structure. Return zero on
```

```
success or ERROR on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */
#include   <stdio.h>  /* for NULL */

#define    ERROR              ( -1)

int getobjptrs( obj, objp)
  unsigned char *obj;
  struct objptrs *objp;
{
  unsigned char *findlenfld( unsigned char *);
  unsigned char *finddatafld( unsigned char *);
  int getobjlen( unsigned char *);
  unsigned char *bytep;
  int objlen;

  objp->tagp = obj;              /* point to the tag field */

  if( ( bytep = findlenfld( obj)) == NULL)    /* found length field? */
          return( ERROR);                     /* no, error */

  objp->lenp = bytep;                         /* mark the length field beginning */

  if( ( objlen = getobjlen( bytep)) < 0)      /* got object length? */
          return( ERROR);                     /* no, error */

  if( ( bytep = finddatafld( bytep)) == NULL)         /* found data field? */
          return( ERROR);                             /* no, error */

  objp->datap = bytep;                                /* mark the data field beginning */

  objp->endobjp = objp->endatap = bytep + objlen;     /* mark the end of the data field */

  if( *objp->lenp == FLDCONMSK)                       /* constructed length data field? */
          objp->endatap -= 2;             /* yes, set the end-of-data pointer accordingly */

  return( 0);
}

// END GETOBJPT.C
```

```
/*---- snmpnettoasc() ----------------------------------------*/
/* Convert an IP address encoded as an SNMP IpAddress object to
a string of ascii characters in the familiar dot notation in a buffer
specified.  Return ERROR on failure; zero on success.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct */
#include   <stdio.h>  /* for sprintf() */

#define    ERROR              (-1)
#define    STRINGSIZ          32

int snmpnettoasc( objptr, buffer, buffsize)
  unsigned char *objptr,
           *buffer;
  int buffsize;
{
  int getobjptrs( unsigned char *, struct objptrs *);
  int getobjlen( unsigned char *);
  long decdsubid( unsigned char **);
  struct objptrs objp;
  int i,j, bytes, objlen;
  unsigned char tempbuf[STRINGSIZ],
                *bufferp,
                *buffend,
                *bytep;

  if( *objptr != IpAddrID)                  /* were we passed the right object? */
            return( ERROR);                         /* no, error */

  if( getobjptrs( objptr, &objp) < 0)       /* got object pointers? */
            return( ERROR - 1);             /* no, error */

  if( ( objlen = getobjlen( objp.lenp)) < 0)    /* got length value? */
          return( ERROR - 2);               /* no, error */

  bufferp = buffer;                         /* point to the buffer beginning */
  buffend = buffer + buffsize;              /* mark the buffer end */
  bytep = objp.datap;

  for( i = 0 ; i < objlen ; i++)
  { /* loop while there are bytes to convert */
    sprintf( tempbuf, "%d.", ( int) decdsubid( &bytep));

    for( bytes = 0 ; tempbuf[bytes] != ( unsigned char) '\0' ; bytes++);
                                            /* decode each ASN.1 integer and convert the integer to an ascii string */

            if( ( int)( buffend - bufferp) <= bytes)    /* is there room in the buffer for the ascii chars? */
            for( j = 0 ; j < bytes ; j++)               /* yes, move them into the buffer */
                    *(bufferp++) = tempbuf[j];
  }

  *(--bufferp) = '\0';                      /* overwrite the last '.' with a '\0' */

  return( ( int)( bufferp - buffer + 1));
}

// END NETTOASC.C
```

```
/*---- setobjlen() ----------------------------------------*/
/* Given the length of an object, encode the length into an ASN.1
object length field.  Return the number of octets used in creating
the length field.  Return ERROR on error.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for FLDCONMSK */
#include   <stdio.h>  /* for 'fprintf()' and NULL */

#define    ERROR              ( -1)

int setobjlen( objlen, lenp)
  int objlen;
  unsigned char *lenp;
{
  if( objlen < 1)                          /* negative or zero length field? */
            return( ERROR);                /* yes, error */

  if( objlen > 127)
  {                    /* use the long form of a length value? */
            int lenlen;
            unsigned char *bytep;

            lenlen = 0;
            lenp += sizeof( objlen);                  /* get the size of the length value */
            bytep = ( unsigned char *)&objlen;        /* point at its least signif byte */

          while( ( int)( bytep - ( unsigned char *) &objlen) < sizeof( objlen) - 1)
          {
                    *( lenp--) = *( bytep++);         /* the last byte of objlen is the last byte of the len fld */
                    lenlen++;                         /* increment the len of the len fld */
          }

          *( lenp--) = *bytep;                        /* the first byte of the objlen is the first byte of the len fld */
          lenlen++;                                   /* increment the len of the len fld */
          *lenp = FLDCONMSK | lenlen;                 /* OR in the number of bytes in the value fld of the len fld */
          return( ++lenlen);                          /* return the number of bytes used in the len fld including the first byte */
  }
  else
  {                    /* use the short form */
            *lenp = ( unsigned char)objlen; /* put the length in the first length byte */
            return( sizeof( *lenp));                  /* return the number of bytes in the len fld */
  }
}

// END SETOBJLN.C
```

```
/*---- getreqid() ----------------------------------------------*/
/* Given a pointer to an object and a pointer to an unsigned long
integer, replace the value of the unsigned long integer with the
value of the object. Return zero on success; return a negative
number on failure.

*/

#include    "../snmp/snmp.h"        /* for objptrs struct, UINTID, etc. */

#define     ERROR                   (-1)                    /* returned error status */

int getreqid( objptr, requestidp)
   unsigned char *objptr;
   long *requestidp;
{
   int getobjptrs( unsigned char *, struct objptrs *);
   int getobjlen( unsigned char *);
   struct objptrs objp;
   unsigned char *endata;
   long *reqidp;
   int len, i;

   if( getobjptrs( objptr, &objp) < 0)          /* got object pointers? */
               return( ERROR);                  /* no, error */

   endata = objp.endatap;                       /* find the end of the data */

   if( *objp.tagp != UINTID)                    /* is the object an integer like the request id should be? */
               return( ERROR - 1);              /* no, error */

   if( ( len = getobjlen( objp.lenp)) < 0)      /* got the length value? */
               return( ERROR - 2);              /* no, error */

   if( len > sizeof( *reqidp))                  /* object too long for a req id? */
               return( ERROR - 3);              /* yes, error */

   reqidp = ( long *)objp.datap;                /* point at the request id */
   *requestidp = *reqidp;                       /* move the value */

   return( 0);                                  /* return the request id */
}

// END GETREQID.C
```

```
/*----- getroutinfo() ----------------------------------------*/
/* Given a network descriptor and a pointer to a router information
structure, return a pointer to the information about the next router
information structure.  Return NULL on error.
*/

#include   <stdio.h>                /* for 'fprintf', NULL */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for routinfo struct */

struct routinfo *getroutinfo( agentaddr, routinfop)
    long agentaddr;
    struct routinfo *routinfop;
{
  extern int univerr;
  extern struct routinfo *firstrinfop;
  int cmpstr( unsigned char *, unsigned char *, int);
  struct routinfo *rinfop;
  unsigned char tempbuf[STRINGSIZ], *bytep;
  int i, j, nxtflag;

  univerr = 0;
  if( ( rinfop = firstrinfop) == NULL)
  {         /* is there any info? */
            univerr = -1;
            return( NULL);                                  /* no, error */
  }
  bytep = ( unsigned char *) &agentaddr;                    /* convert agentaddr to string */

  for( i = 0, j = 0 ; ( i < sizeof( agentaddr)) && ( j < STRINGSIZ) ; i++, bytep++)
  {

          if( ( j >= STRINGSIZ) && ( i < sizeof( agentaddr)))
          {
            univerr = -2;                               /* was there enough buffer space? */
            return( NULL);                              /* no, error */
          }
  }
  tempbuf[j-1] = '\0';

  if( routinfop == NULL)                    /* find first matching block? */
            nxtflag = 1;                    /* yes */
  else
            nxtflag = 0;                    /* no */
  do
  {
           i = cmpstr( ( unsigned char *) &tempbuf[0], ( unsigned char *) &( rinfop->routeraddr[0]), j);
           if( i != 0)              /* did the compare fail? */
               rinfop = rinfop->nxtblock;   /* yes, go to the next block */
          else
          {                                 /* the compare succeeded */
               if( nxtflag <= 0)
               {                    /* are we looking for this block? */
                        i = 1;                          /* no, negate the compare */
                        if( routinfop == rinfop)        /* looking for next block? */
                        nxtflag = 1;                    /* yes, set the flag */
                        rinfop = rinfop->nxtblock;      /* point to next block */
               }       /* else, we are looking for this block, exit the loop */
          }
  } while( ( i != 0) && ( rinfop != NULL));

  if( i != 0)
  {                     /* did we find the right info? */
            univerr = -3;
            return( NULL);                              /* no, error */
  }
  return( rinfop);
}
// END GETRINFO.C

/*----- getvarinfo() ----------------------------------------*/
/* Given a pointer to a router information block, a pointer to an
```

```
object info block, a pointer to a null terminated string which
contains a encoded index value, and the length of the string, return a
pointer to the matching variable information block.  Return NULL on
error.

*/

#include   <stdio.h>                                    /* for 'fprintf', NULL */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"            /* for routinfo struct */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for objinfo struct */

struct varinfo *getvarinfo( routinfop, objinfop, indexp, indexlen)
    struct routinfo *routinfop;
    struct objinfo *objinfop;
    unsigned char *indexp;
    int indexlen;
{
  extern int univerr;
  int cmpstr( unsigned char *, unsigned char *, int);
  struct varinfo *vblkp;
  int i,len;

  if( ( vblkp = routinfop->vblockp) == NULL)
  {       /* are there variable info blocks? */

            univerr = -1;
            return( NULL);                                 /* no, error */
  }
                  /* find the length of the variable string */

  for( len = 0 ; objinfop->objname[len] != '\0' ; len++);
            len++;                                       /* add in the '\0' too */

  do
  {       /* loop while a matching variable info block isn't found */
          if( cmpstr( ( unsigned char *) &( vblkp->variable[0]), ( unsigned char *) objinfop->objname, len) == 0)
          {
                                  /* variable names match? */
                  if( *indexp != '\0')
                  {       /* is the index a zero? */
                          if( indexlen == ( int) vblkp->indexlen)
                          {       /* no, index lengths match? */


                                  i = cmpstr( indexp, ( unsigned char *) &( vblkp->index[0]),
                                                  ( int) vblkp->indexlen);

                                    /* yes, compare the index strings ( match makes i = 0) */
                          }
                          else
                          {       /* index length didn't match */
                             i = (-1);
                          }
                  }
                  else
                  {       /* the index is a zero, i.e., there is no index */

                          if( vblkp->indexlen != 0)
                          {       /* is this variable's index a zero, too? */
                                  i = ( -1);              /* no, no variable match */
                          }
                          else                            /* this variable's index is zero too */
                                  i = 0;                  /* found a match */
                  }
          }
          else
          {                                               /* variable names didn't match */
            i = (-1);
          }
          if( i != 0)                                     /* found a matching variable info block? */
            vblkp = vblkp->nxtblock;                      /* no, go to the next one */
```

```
    } while( ( vblkp != NULL) && ( i != 0));

    if( i != 0)
    {                   /* was a match found? */
                univerr = -2;
                return( NULL);                                  /* no, error */
    }

    return( vblkp);
}

// END GETVINFO.C
```

```
/*---- ifIndex_f() ----------------------------------------*/
/* Determine the index number of the interface about which
"interesting data" is being reported.  Return that number or return
ERROR on error.

*/

#include   <stdio.h>                                    /* for 'fprintf' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"            /* for objptrs struct, USEQID, etc. */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for objinfo struct */

#define    ERROR              (-1)                     /* returned error status */

int ifIndex_f( obj)
  unsigned char *obj;
{
  unsigned char *getnxtobj( unsigned char *);
  int getintval( unsigned char *, void *, int);
  int getobjptrs( unsigned char *, struct objptrs *);
  struct objptrs objp;
  int status;
  static int i;
  unsigned char *bytep;

  if( *obj != USEQID)
  {
            fprintf( stderr, "ifIndex: given object not a sequence\n");
            return( ERROR);
  }

  if( getobjptrs( obj, &objp) < 0)
  {                    /* got object pointers? */
            fprintf( stderr, "ifIndex: couldn't get original object pointers\n");
            return( ERROR - 1);
  }

  if( ( bytep = getnxtobj( objp.datap)) == NULL)
  {       /* got the object after the object identifier object? */

            fprintf( stderr, "ifIndex: couldn't find the integer object\n");
            return( ERROR - 2);
  }

  if( *bytep != UINTID)
  {       /* is this an integer object? */
            fprintf( stderr, "ifIndex: object wasn't an integer\n");
            return( ERROR - 3);                        /* no, error */
  }

  if( ( status = getintval( bytep, ( void *) &i, sizeof( i))) < 0)
  {
                                                       /* got the integer value? */
            fprintf( stderr, "ifIndex: couldn't get integer value, status was %d\n", status);
            return( ERROR - 4);                        /* no, error */
  }
  return( i);                                          /* yes, return it */
}

// END IFINDEX.C
```

```
/*----- ifOperStatus strings ----------------------------------*/
char ifopstat[][10] = { "", "up", "down", "testing", "" };

/*----- ifOperStatus_f() --------------------------------------*/
/* Perform the necessary functions when a response from an SNMP agent
contains information about an interfaces operational status.  Return
ERROR on error.

*/

#include   <stdio.h>              /* for 'fprintf' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"        /* for objptrs struct, USEQID, etc. */
#include   "\gary\snmp\snmp_src\mib\mib.h"          /* for objinfo struct */

#define    ERROR                  (-1)             /* returned error status */

int ifOperStatus_f( seqp)
  unsigned char *seqp;
{
  extern struct objinfo *isoorgroot;
  struct objinfo *searcharr( struct objinfo *, char *);
  extern char ifopstat[][10];
  extern int univerr;
  unsigned char *getnxtobj( unsigned char *);
  int getintval( unsigned char *, void *, int);
  int getobjptrs( unsigned char *, struct objptrs *);

  int interface;
  static int index;
  struct objptrs objp;
  struct objinfo *objinfop;
  unsigned char *endata,
                *bytep;
  univerr = 0;

  if( *seqp != USEQID)
  {                    /* were we passed a sequence? */
               univerr = -1;
               return( ERROR);                    /* no, error */
  }

  if( getobjptrs( seqp, &objp) < 0)
  {         /* got object pointers? */

              univerr = -2;
              return( ERROR - 1);                          /* no, error */
  }

  if( *objp.datap != UOBJIDID)
  {         /* is the first obj an obj id? */
              univerr = -3;
              return( ERROR - 2);                          /* no, error */
  }
  endata = objp.endatap;                    /* mark the end of data */

  if( getobjptrs( objp.datap, &objp) < 0)
  {         /* got obj id pointers? */
              univerr = -4;
              return( ERROR - 3);                          /* no, error */
  }

  if( (objp.endatap - objp.datap) != 10)
  {         /* obj id wrong size? */
              univerr = -5;
              return( ERROR - 4);                          /* yes, error */
  }

  interface = ( int)( *( objp.endatap - 1));      /* get the interface no. */

  if( ( bytep = getnxtobj( objp.tagp)) == NULL)
  {         /* got next obj? */
```

```
            univerr = -6;
            return( ERROR - 5);                                 /* no, error */
    }

    if( ( objinfop = searcharr( isoorgroot, "ifOperStatus")) == NULL)
    {
            fprintf( stderr, "ifOperStatus_f: can't find ifOperStatus structure\n");
            univerr = -7;
            return( ERROR - 6);
    }

    if( *bytep != ( unsigned char) objinfop->type)
    {       /* is the obj the right type (should be UINTID)? */
            univerr = -8;
            return( ERROR - 7);                                 /* no, error */
    }

    if( getintval( bytep, ( void *) &index, sizeof( index)) < 0)
    {
                                                        /* got status index value? */
            univerr = -9;
            return( ERROR - 8);                                 /* no, error */
    }

    fprintf( stderr, "ifOperStatus_f: interface %d is %s\n", interface,
                                                                ifopstat[index]);
    return( index);
}

// END IFOPSTAT.C
```

```
/*---- resperr() ----------------------------------------------*/
/* Given the error status, error index, and a pointer to the variable
bindings of a response packet, analyze the error.

*/

#include   <stdio.h>              /* for 'fprintf' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"      /* for objptrs struct, USEQID, etc. */
#include   "\gary\snmp\snmp_src\mib\mib.h"        /* for objinfo struct */
#include   "\gary\snmp\snmp_src\net\net.h"        /* for sockdata struct */

#define    ERROR               (-1)               /* returned error status */

int resperr( socketp, errstat, errindex, varbindp)
  struct sockdata *socketp;
  int errstat, errindex;
  unsigned char *varbindp;
{
  int getobjptrs( unsigned char *, struct objptrs *);
  unsigned char *getnxtobj( unsigned char *);
  struct objinfo *getobjidstrc( unsigned char *);
  long decdsubid( unsigned char **);

  static char error0[] = "noError";
  static char error1[] = "tooBig";
  static char error2[] = "noSuchName";
  static char error3[] = "badValue";
  static char error4[] = "readOnly";
  static char error5[] = "genErr";
  static char *(err_p[]) = { error0, error1, error2, error3, error4, error5 } ;
  struct objinfo *objinfop, *objinfop1;

  struct objptrs objp;
  unsigned char *bytep, *indexp;
  int i, indexlen;

  if( errstat == 0)
            return( ERROR);

  if( errstat == 1)
  {
            fprintf( stderr, "Agent %s, ", &( socketp->routinfop->routeraddr[0]));
            fprintf( stderr, "request packet was too large\n");
            return( 0);
  }

  if( errstat == 5)
  {
            fprintf( stderr, "Agent %s, ", &( socketp->routinfop->routeraddr[0]));
            fprintf( stderr, "general error\n");
            return( 0);
  }

  if( *varbindp != USEQID)                    /* were we given a sequence? */
            return( ERROR);                   /* no, error */

  if( getobjptrs( varbindp, &objp) < 0)       /* got the pointers for the seq? */
            return( ERROR - 1);               /* no, error */

  bytep = objp.datap;                         /* point at the first sequence in the variable bindings */

  for( i = 1 ; i < errindex ; i++)
  {         /* while this isn't the bad sequence loop */

            if( ( bytep = getnxtobj( bytep)) == NULL)  /* got a pointer to the   next sequence? */
            return( ERROR - 2);                        /* no, error, abort and exit */
  }                 /* bytep now points to the offending sequence */

  if( getobjptrs( bytep, &objp) < 0)          /* got pointers for the seq? */
            return( ERROR - 3);               /* no, error */
```

```
if( ( objinfop = getobjidstrc( objp.datap)) == NULL)
{
        fprintf( stderr, "Agent %s\n", &( socketp->routinfop->routeraddr[0]));
        fprintf( stderr, "\trequest was bad, here is the offending object\n");

        for( i = 0 ; i < ( int)( objp.endatap - objp.datap) ; i++)
  {
          if( i % 25 == 0)
                  putc( '\n', stderr);

          fprintf( stderr, "%02x ", objp.datap[i]);
        }
        putc( '\n', stderr);

        putc( '\n', stderr);
        return( 0);
}

if( getobjptrs( objp.datap, &objp) < 0)
{       /* got pointers to obj id? */
        fprintf( stderr, "Agent %s\n", &( socketp->routinfop->routeraddr[0]));
        fprintf( stderr, "\trequest was bad, here is the offending object\n");
        for( i = 0 ; i < ( int)( objp.endatap - objp.tagp) ; i++)
        {
                if( i % 25 == 0)
                putc( '\n', stderr);
                fprintf( stderr, "%02x ", objp.datap[i]);
        }

        putc( '\n', stderr);
        putc( '\n', stderr);
        return( 0);                     /* no, error */
}
                /* find the beginning of the index in the obj id */
objinfop1 = objinfop;
i = 1;

do
{               /* go up the structure tree finding how many levels we move up */
        while( objinfop1->flag > (-1))  /* find first structure in this group */
                objinfop1--;

        if( objinfop1->flag == (-1))
        {       /* from the first structure find the one pointing to this group */

                objinfop1 = ( struct objinfo *) objinfop1->ptr;
                i++;                    /* increment the number of levels traversed */
        }
} while( objinfop1->flag > (-16));      /* loop while we aren't at the top of the structure tree */

                /* i + 5 now is the number of bytes into the obj id the index starts */

indexp = objp.datap + i + 5;            /* point to the index */
indexlen = objp.endatap - indexp;       /* get the index length */

fprintf( stderr, "Agent %s doesn't like %s", &( socketp->routinfop->routeraddr[0]), &( objinfop->objname[0]));

while( indexp < objp.endatap)
        fprintf( stderr, ".%d", ( int) decdsubid( &indexp));

fprintf( stderr, "\n\terror %s\n", err_p[errindex]);
return( 0);
}

// END RESPERR.C
/*---- saveval() ----------------------------------------------*/
/* Given a pointer to a variable information block and a pointer to
an ASN.1 object, put the value of the object in the variable info block.
Return zero on success; ERROR on error.

*/
```

```
#include   <stdio.h>                                  /* for 'fprintf' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"          /* for objptrs struct, USEQID, etc. */
#include   <sys/types.h>                              /* for 'ftime()' */
#include   <sys/timeb.h>                              /* for 'ftime()' */

#define    ERROR              (-1)                    /* returned error status */

int saveval( vblkp, valuep)
  struct varinfo *vblkp;
  unsigned char *valuep;
{
  int getintval( unsigned char *, void *, int);
  int getobjlen( unsigned char *);
  int getobjptrs( unsigned char *, struct objptrs *);
  int cmpstr( unsigned char *, unsigned char *, int);
  void ftime( struct timeb *);

  int i;
  long li;
  struct objptrs objp;
  unsigned char *bytep;

  struct timeb  timeinfo;

  if( *valuep == UOCTSTRID)
  {
          if( ( ( unsigned char) vblkp->vartype != UOCTSTRID) && ( ( unsigned char) vblkp->vartype != DisplStrID) &&
                                                                ( ( unsigned char) vblkp->vartype != IntegerStrID))
                  return( ERROR);
  }
  else
  {
          if( ( unsigned char) vblkp->vartype != *valuep)       /* type of returned value matches type of variable? */
            return( ERROR);             /* no, error */
  }

  if( getobjptrs( valuep, &objp) < 0)                   /* got object pointers? */
          return( ERROR - 1);

  switch( vblkp->vartype)
  {       /* find out how much storage is available for the returned value */

          case ( UINTID):               /* VTYPEINT (2) bytes used for value */
                          if( getintval( valuep, ( void *) &i, sizeof( i)) < 0)
                                                        /* got integer value? */
                                  return( ERROR - 8);   /* no, error */
                          if( *( ( int *) vblkp->varvalue) != i)
                          {       /* is this a new value? */

                                  *( ( int *) vblkp->varvalue) = i;     /* yes, save the returned value */

                                  vblkp->changeflag++;                  /* set the change flag */
                          }
                          break;

        case ( UOCTSTRID):              /* VTYPESTR (32) bytes used for value */
        case ( IntegerStrID):
                                  /* move the value string to the storage area */

                          if( ( objp.endatap - objp.datap) > STRINGSIZ)          /* string too large? */
                                  return( ERROR - 9);                           /* yes, error */
                                          /* point to data save area */
                          bytep = ( unsigned char *) vblkp->varvalue + sizeof( int);

                          if( ( int)( objp.endatap - objp.datap) != *( ( int *) vblkp->varvalue))
                          {
                                          /* string lengths different? */
                                          /* yes, put new one in */
                                  for( i = 0 ; i < ( objp.endatap - objp.datap) ; i++)
                                          bytep[i] = objp.datap[i];
```

```
                        *( ( int *) vblkp->varvalue) = i;
                        vblkp->changeflag++;
                }
                else
                {
                        if( cmpstr( bytep, objp.datap, ( objp.endatap - objp.datap)) != 0)
                        {
                                for( i = 0 ; i < ( objp.endatap - objp.datap) ; i++)
                                        bytep[i] = objp.datap[i];
                                vblkp->changeflag++;
                        }
                }
                break;

case ( DisplStrID):     /* Display String (octal string) variable */
                        /* move the value string to the storage area */

                if( ( objp.endatap - objp.datap) > DISPSTRSIZ)

                                        /* string too large? */
                return( ERROR - 2);     /* yes, error */

                                /* point to data save area */
                bytep = ( unsigned char *) vblkp->varvalue + sizeof( int);

                if( ( int)( objp.endatap - objp.datap) != *( ( int *) vblkp->varvalue))
                {
                                /* string lengths different? */
                                /* yes, put new one in */

                        for( i = 0 ; i < ( objp.endatap - objp.datap) ; i++)
                                bytep[i] = objp.datap[i];

                        *( ( int *) vblkp->varvalue) = i;
                        vblkp->changeflag++; /* increment change flag */
                }
                else
                {               /* string lengths are the same */
                        if( cmpstr( bytep, objp.datap, ( objp.endatap - objp.datap))!= 0)
                        {
                                /* strings different? yes, replace old with new */

                                for( i = 0 ; i < ( objp.endatap - objp.datap) ; i++)
                                        bytep[i] = objp.datap[i];

                                vblkp->changeflag++; /* increment change flag */
                        }
                }
                break;
case ( IpAddrID):       /* VTYPEIPAD (16) bytes used for value */
                if( getobjlen( objp.lenp) != 4)
                {
                        return( ERROR - 10);
                }

                for( i = 0 ; i < 4 ; i++)
                        ( ( unsigned char *) vblkp->varvalue)[i] = objp.datap[i];
                break;
case ( CounterID):      /* VTYPECTR (4) bytes used for value */
                if( ( i = getintval( valuep, ( void *) &li, sizeof( li))) < 0)
                {
                                        /* got integer value? */
                        fprintf( stderr, "saveval: getintval returned %d\n", i);
                        return( ERROR - 11);            /* no, error */
                }
                if( *( ( long *)vblkp->varvalue) != li)
                {       /* is this a new value? */
                        *( ( long *)vblkp->varvalue) = li;              /* yes, save the returned value */
                        vblkp->changeflag++;
                }
                break;
```

```
        case ( GaugeID):            /* VTYPEGAGE (4) bytes used for value */

                                    if( getintval( valuep, ( void *) &li, sizeof( li)) < 0)
                                                                        /* got integer value? */
                                            return( ERROR - 12);        /* no, error */
                                    if( *( ( long *)vblkp->varvalue) != li)
                                    {       /* is this a new value? */
                                            *( ( long *)vblkp->varvalue) = li;  /* save the returned value */
                                            vblkp->changeflag++;
                                    }
                                    break;

        case ( TimeTicksID):        /* VTYPETIME (4) bytes used for value */

                                    if( getintval( valuep, ( void *) &li, sizeof( li)) < 0)
                                                                        /* got integer value? */
                                            return( ERROR - 13);        /* no, error */
                                    if( *( ( long *)vblkp->varvalue) != li)
                                    {       /* is this a new value? */

                                            *( ( long *)vblkp->varvalue) = li;  /* yes, save the returned value */
                                            vblkp->changeflag++;
                                    }
                                    break;

        case ( UOBJIDID):           /* object identifier object */
                                    /* move the value string to the storage area */

                                    if( ( objp.endatap - objp.datap) > VTYPESTR) /* string too large? */
                                            return( ERROR - 3);         /* yes, error */

                                                /* point to data save area */
                                    bytep = ( unsigned char *) vblkp->varvalue + sizeof( int);
                                    if( ( int)( objp.endatap - objp.datap) != *( ( int *) vblkp->varvalue))
                                    {
                                                /* string lengths different? */
                                                        /* yes, put new one in */
                                            for( i = 0 ; i < ( objp.endatap - objp.datap) ; i++)
                                                    bytep[i] = objp.datap[i];
                                            *( ( int *) vblkp->varvalue) = i;
                                            vblkp->changeflag++; /* increment change flag */
                                    }
                                    else
                                    {               /* string lengths are the same */

                                            if( cmpstr( bytep, objp.datap, ( objp.endatap - objp.datap)) != 0)
                                            {
                                            /* strings different? yes, replace old with new */
                                                    for( i = 0 ; i < ( objp.endatap - objp.datap) ; i++)
                                                            bytep[i] = objp.datap[i];
                                                    vblkp->changeflag++; /* increment change flag */
                                            }
                                    }
                                    break;
        case ( OpaqueID):   /* unknownd storage requirements */
                                    return( ERROR - 14);
        default:            /* not implemented yet */

          return( ERROR - 15);
    }

    ftime( &timeinfo);              /* get the time of the change */

    vblkp->timestamp = timeinfo.time;       /* save it */

    return( 0);
}

// END SAVEVAL.C
```

```
/*---- response error status strings ----------------------------*/
char errstatp[][12] = { "noError", "tooBig", "noSuchName", "badValue",
                        "readOnly", "genErr", "" };

/*---- analresp() -----------------------------------------------*/
/*  Given a response protocol data unit (pdu), analyze it.  Return
negative numbers on error, the network descriptor on no error.


*/

#include   <stdio.h>              /* for 'fprintf' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"      /* for objptrs struct, USEQID, etc. */
#include   "\gary\snmp\snmp_src\mib\mib.h"        /* for objinfo struct */
#include   "\gary\snmp\snmp_src\net\net.h"        /* for sockdata struct */

#define    ERROR               (-1)               /* returned error status */

int analresp( nd, pdu)
  int nd;
  unsigned char *pdu;
{
  extern char errstatp[][12];
  extern int univerr;
  extern struct sockdata *socketroot;
  int getobjptrs( unsigned char *, struct objptrs *);
  int getintval( unsigned char *, void *, int);
  extern int getreqid( unsigned char *, long *);
  int saveval( struct varinfo *, unsigned char *);
  struct varinfo *getvarinfo( struct routinfo *, struct objinfo *, unsigned char *, int);
  unsigned char *getnxtobj( unsigned char *);
  struct objinfo *getobjidstrc( unsigned char *);
  int resperr( struct sockdata *, int, int, unsigned char *);

  struct sockdata *sockdatap;
  struct objinfo *objinfop;
  struct objptrs objp;
  struct varinfo *vblkp;

  unsigned char *bytep, *objidp, *objseqp,  *endata, *indexp;
  int i, indexlen, errstatus;
  long requestid;
                    /* decode the PDU implied sequence */

  univerr = 0;

  if( *pdu != RESPID)                    /* pointing to a response pdu? */
            return( ERROR);                          /* no, return error */

  if( getobjptrs( pdu, &objp) < 0) /* got object pointers? */
            return( ERROR - 1);                              /* no, error */

  endata = objp.endatap;                 /* find the end of the data */

  if( getreqid( objp.datap, &requestid) != 0)  /* got the request id value? */
            return( ERROR - 2);                              /* no, error */

  if( ( sockdatap = socketroot) == NULL)     /* no outstanding responses? */
            return( ERROR - 3);                              /* right, error */
  while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
  {
            sockdatap = sockdatap->nxtstrcp;
  }

  if( sockdatap->reqid != requestid)
  {         /* is this one of the packets we want? */

            fprintf( stderr, "analresp: received request id = %08lx,\n", requestid);
    fprintf( stderr, "\tsent request id = %08lx\n", sockdatap->reqid);
    return( ERROR - 4);                              /* no, error */
  }
```

```
if( ( bytep = getnxtobj( objp.datap)) == NULL)          /* can we get the
                                                           next object (err-stat)? */
        return( ERROR - 5);                             /* no, error */

if( getobjptrs( bytep, &objp) < 0)              /* got err-stat object pointers? */
        return( ERROR - 6);                             /* no, error */

if( *objp.tagp != UINTID)                       /* is the object an integer like
                                                   the error-status should be? */
        return( ERROR - 7);                             /* no, error */

if( getintval( objp.tagp, ( void *) &i, sizeof( i)) < 0)
                                                        /* got value of error-status? */
        return( ERROR - 8);                             /* no, error */

if( ( i > 5) || ( i < 0))
{               /* error-status valid? */
        fprintf( stderr, "analresp: error status on response was %d\n", i);
        return( ERROR - 9);                             /* no, error */
}

errstatus = i;                  /* save the error status */

if( ( bytep = getnxtobj( objp.tagp)) == NULL)           /* can get the next object (err-index)? */
        return( ERROR - 10);                            /* no, error */

if( getobjptrs( bytep, &objp) < 0)              /* got object pointers? */
        return( ERROR - 11);                            /* no, error */

if( *objp.tagp != UINTID)                       /* is the object an integer like the error-index should be? */
        return( ERROR - 12);                            /* no, error */

if( getintval( objp.tagp, ( void *) &i, sizeof( i)) < 0)
                                                /* got value of error-index? */
        return( ERROR - 13);                            /* no, error */

if( errstatus > 0)
{       /* error being reported by the device? */

        univerr = i;            /* yes, save the index */

        if( ( bytep = getnxtobj( objp.tagp)) == NULL)
        {       /* got the variable bindings object? */

                fprintf( stderr, "analresp: error status is %d, error index is %d\n", errstatus, i);            /* no, print error */
                fprintf( stderr, "\tThe response packet is\n");
                for( bytep = pdu ; bytep < endata ; bytep++)
                {
                    if( ( ( int)( bytep - pdu) % 25) == 0)
                    putc( '\n', stderr);
                    fprintf( stderr, "%02x ", *bytep);
                }
                putc( '\n', stderr);
        }
        else
        {               /* got variable bindings object */
                if( resperr( sockdatap, errstatus, i, bytep) < 0)
                {       /* analyze the error okay? */

                        fprintf( stderr, "analresp: error status is %d, error index is %d\n",  errstatus, i);
                                                                                        /* no, print error */
                        fprintf( stderr, "\tThe response packet is\n");

                        for( bytep = pdu ; bytep < endata ; bytep++)
                        {
                                if( ( ( int)( bytep - pdu) % 25) == 0)
                                        putc( '\n', stderr);
                                fprintf( stderr, "%02x ", *bytep);
                        }
                        putc( '\n', stderr);
```

```
                }
        }
        return( ERROR - 14); /* exit without analyzing the packet further */
}

if( ( bytep = getnxtobj( objp.tagp)) == NULL)          /* got the next object (var-bindings)? */
        return( ERROR - 15);                            /* no, error */

if( getobjptrs( bytep, &objp) < 0)          /* got object pointers? */
        return( ERROR - 16);                            /* no, error */

if( *objp.tagp != USEQID)                   /* is the object a sequence like the var-bindings should be? */
        return( ERROR - 17);                            /* no, error */

objseqp = objp.datap;           /* mark the first object id seq beginning */

while( objseqp < endata)
{
        struct objinfo *objinfop1;
        if( *objseqp != USEQID)                         /* found the obj id sequence? */
          return( ERROR - 18);                          /* no, error */

        if( getobjptrs( objseqp, &objp) < 0)            /* got object pointers? */
          return( ERROR - 19);                          /* no, error */

        if( *objp.datap != UOBJIDID)                    /* is the first obj an obj id? */
          return( ERROR - 20);                          /* no, error */
                /* time to find out what the packet says */

        if( ( objinfop = getobjidstrc( objp.datap)) == NULL)            /* is the object in the MIB? */
                return( ERROR - 21);                                    /* no, error */

        if( getobjptrs( objp.datap, &objp) < 0)                 /* get pointers to obj id? */
          return( ERROR - 22);                                  /* no, error */

                /* find the beginning of the index in the obj id */

        objinfop1 = objinfop;
        i = 1;

        do
        {               /* go up the structure tree finding how many levels we move up */
                while( objinfop1->flag > (-1))  /* find first structure in this group */
                        objinfop1--;

                if( objinfop1->flag == (-1))
                {       /* from the first structure find the one pointing to this group */

                        if( ( objinfop1 = ( struct objinfo *) objinfop1->ptr) == NULL)
                        {
                                fprintf( stderr, "analresp: MIB is faulty, can't backup\n");
                                return( ERROR - 23);
                        }
                        i++;                    /* increment the number of levels traversed */
                }

      } while( objinfop1->flag > (-16)); /* loop while we aren't at the top of the structure tree */

                /* i + 1 now is the number of bytes into the obj id the    index starts */

      indexp = objp.datap + i + 1;                      /* point to the index */
      indexlen = objp.endatap - indexp;  /* get the index length */

      if( ( vblkp = getvarinfo( sockdatap->routinfop, objinfop, indexp,    indexlen)) == NULL)
      {       /* is there a variable block with the correct variable name and index? */

              fprintf( stderr, "analresp: getvarinfo() returned error %d\n",          univerr);
              return( ERROR - 24);              /* no, error */
      }

      if( ( indexp = getnxtobj( objp.tagp)) == NULL)
```

```
        {       /* got the pointer to the value object? */

                fprintf( stderr, "analresp: couldn't get value pointer, error %d\n",    univerr);
                return( ERROR - 25); /* no, error */
        }

        if( ( i = saveval( vblkp, indexp)) < 0)
        { /* was the value saved? */

                fprintf( stderr, "analresp: saveval() returned error %d\n", i);
                return( ERROR - 26);            /* no, error */
        }

        if( objinfop->funct != NULL)
        {       /* is there a function to be called to process this variable? */

                if( objinfop->funct( objseqp) < 0)  /* yes, did it run okay? */
                        fprintf( stderr, "analresp: function %s_f returned error %d\n",     objinfop->objname, univerr);
        }

        if( ( objseqp = getnxtobj( objseqp)) == NULL)           /* got next seq? */
          return( ERROR - 27);                                  /* no, error */
  }
  return( sockdatap->nd);
}
```

// END ANALRESP.C

```
/*---- getobjidstrc() ----------------------------------------*/
/* Given a pointer to an object identifier object, return a pointer
to the MIB structure associated with that object.  Return NULL
on error.

*/

#include   <stdio.h>                                    /* for 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"            /* for UOBJIDID, etc. */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for objinfo struct */

struct objinfo *getobjidstrc( obj)
    unsigned char *obj;
{
  extern struct objinfo *isoorgroot;
  extern unsigned char isoorgval;
  int getobjptrs( unsigned char *, struct objptrs *);
  unsigned char *bytep;
  int i;
  struct objinfo *objinfop;
  struct objptrs objp;

  if( *obj != UOBJIDID)
  {                    /* pointing to an obj. id.? */
          fprintf( stderr, "getobjidstrc: not an obj id object\n");
          return( NULL);                                    /* no, error */
  }

  if( getobjptrs( obj, &objp) < 0)                          /* got object pointers? */
          return( NULL);                                    /* no, error */

  bytep = objp.datap;                                       /* point to the data */

  if( *bytep != isoorgval)
  {
          fprintf( stderr, "\ngetobjidstrc: header value is wrong\n");
          fprintf( stderr, "\tisoorgval = %02x, *bytep = %02x\n",    isoorgval, *bytep);
          return( NULL);
  }

  objinfop = isoorgroot;                    /* start at the iso-org structure */
  objinfop++;                               /* point to the first valid array entry */
  bytep++;                                  /* point at the next obj id byte */

  for( i = 1 ; ( i < ( int)*bytep) && ( !( ( objinfop->objname == NULL) &&      ( objinfop->flag == ( -1)))); i++, objinfop++);
                    /* move down the array of structures until the proper one is found or the end of the array is reached */

    if( ( objinfop->objname == NULL) && ( objinfop->flag == ( -1)))
    {
                                        /* didn't find a match? */
          fprintf( stderr, "\ngetobjidstrc: first pointer out of bounds\n");
          return( NULL);
    }

    bytep++;                            /* increment the obj id byte pointer */

    while( ( objinfop->flag != 0) && ( bytep < objp.endatap))
    {
                    /* loop while there are more sub-structures and there are more bytes in the object identifier */

              objinfop = ( struct objinfo *) objinfop->ptr;         /* point to the next array of structures */
              objinfop++;                                           /* point to the first valid array entry */

              for( i = 1 ; ( i < ( int)*bytep) && ( !( ( objinfop->objname == NULL)
                                                          && ( objinfop->flag == ( -1)))); i++, objinfop++);

                  /* move down the array of structures until the proper one is found or the end of the array is reached */

                  if( ( objinfop->objname == NULL) && ( objinfop->flag == ( -1)))
                  {
                                        /* didn't find a match? */
```

```
                        fprintf( stderr, "\ngetobjidstrc: pointer out of bounds\n");
                        return( NULL);
                }
                bytep++;                        /* increment the obj pointer */
        }

        if( objinfop->flag != 0)
        {       /* didn't finish walking the structures? */

                fprintf( stderr,"getobjidstrc: didn't finish walking the structures\n");
                return( NULL);                  /* right, error */
        }
        return( objinfop); /* return the pointer to the structure associated with this object identifier */
}


// END GETOBJID.C
```

```
/*---- getreq() ----------------------------------------*/
/* Given a pointer to a structure which contains information about what
SNMP variables values should be retrieved from which router,
and a packet buffer of 'buflen' bytes, create an SNMP packet
containing a Get Request for the value of that variable. Return the
integer used in the packet as the request identifier value on
successfully formating the packet, or ERROR on error.

*/

#include   <stdio.h>                                        /* for 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"                /* for routinfo struct */
#include   "\gary\snmp\snmp_src\mib\mib.h"          /* for objinfo struct */

#define    ERROR            (-1)                           /* error returned status */

long getreq( routinfobp, buffer, buflen)
   struct routinfo *routinfobp;
   unsigned char *buffer;
   int *buflen;
{
  extern struct objinfo *isoorgroot
  extern int univerr;

  struct objinfo *searcharr( struct objinfo *, char *);
  int movenstr( unsigned char *, unsigned char *);
  int rand( void);
  int getobjptrs( unsigned char *, struct objptrs *);
  unsigned char *getvardesc( struct objinfo *);
  int setobjlen( int, unsigned char *);
  int atoi( const char *);

  struct objinfo *objinfop;
  struct varinfo *vblkp;
  unsigned char *bufferp,
                *buffend,
                *pdulenp,
                *seqlenp,
                *reqlenp,
                *varblenp,
                *bytep;

  long *reqidp;
  struct objptrs objp;
  int status;

  univerr = 0;
  vblkp = routinfobp->vblockp;
  buffend = buffer + *buflen;              /* point to the end of the buffer */
  bufferp = buffer;                        /* point to the beginning of the buffer */

  if( buffend <= bufferp + 53)
  {       /* is there enough room for a */
          univerr = -1;                                          /* get request header? */
          return( ( unsigned long) ERROR);                       /* no, error */
  }

  *( bufferp++) = USEQID;                  /* put in the sequence identifier byte */

  pdulenp = bufferp++;                     /* mark and skip the length field */

  *( bufferp++) = UINTID;                  /* put in the verison id */
  *( bufferp++) = 1;                       /* put in the version length */
  *( bufferp++) = SNMPVER;                 /* put in the version number */
  *( bufferp++) = UOCTSTRID;               /* put in the community id */
                                           /* get the community string's length */


   for( status = 0, bytep = ( unsigned char *) &( routinfobp->community[0]) ;
          *bytep != '\0' ; status++, bytep++);
   if( status < 1)
```

```
{                       /* is the community a null string? */
            univerr = -2;                       /* yes, error */
            return( ( unsigned long) ERROR);
}
                                                /* set the community field length */
if( ( status = setobjlen( status, bufferp)) < 0)
{       /* successful? */
            univerr = -3;                       /* no, error */
            return( ( unsigned long) ERROR);
}

if( status > 1)
{                       /* object length too long for length field? */
            univerr = -4;                       /* yes, error */
            return( ( unsigned long) ERROR);
}

bufferp += status;              /* point past the community len field */

if( ( status = movenstr( ( unsigned char *) &( routinfobp->community[0]), bufferp)) < 1)
{
                                                /* did we enter the community? */
            univerr = -5;                       /* no, error */
            return( ( unsigned long) ERROR);
}

bufferp += status;              /* point past the community data field */
*( bufferp++) = GETREQID;                       /* put in the Get Request Identifier */
reqlenp = bufferp++;                            /* mark and skip the GetReq length */
*( bufferp++) = UINTID;                         /* put in this request's identifer identifier */

if( ( status = setobjlen( sizeof( *reqidp), bufferp)) < 0)
{
            univerr = -6;
            return( ( unsigned long) ERROR);
}

if( status > 1)
{                       /* object length too long for length field? */
            univerr = -7;                       /* yes, error */
            return( ( unsigned long) ERROR);
}

bufferp += status;              /* point past the request id len field */
reqidp = ( long *)bufferp;                      /* get a random number and move */
*reqidp = ( long) rand();                       /* it into the request id field */

        /* note: rand() generates a number >= 0 so the request id is always

positive.  save *reqidp for successful return status */
bufferp += sizeof( *reqidp);                    /* point past the request id field */

*( bufferp++) = UINTID;                         /* put in the error status id */
*( bufferp++) = 1;                              /* put in the error status length */
*( bufferp++) = 0;                              /* put in the error status value */
*( bufferp++) = UINTID;                         /* put in the error index id */
*( bufferp++) = 1;                              /* put in the error index length */
*( bufferp++) = 0;                              /* put in the error index value */
*( bufferp++) = USEQID;                         /* put in the variable bindings id */

varblenp = bufferp++;                           /* mark and skip the variable bindings len */



        while( vblkp != NULL)
        {
                unsigned char *seqbegp;
                seqbegp = bufferp;              /* mark the beginning of this sequence */
                *( bufferp++) = USEQID;         /* put in the first object id sequence id */
                seqlenp = bufferp++;            /* mark and skip the sequence length field */
```

```
            if( ( objinfop = searcharr( isoorgroot, ( char *) &( vblkp->variable[0]))) == NULL)
            {
                                /* found the variable name in the database? */
                    univerr = -8;
                    return( ( unsigned long) ERROR);            /* no, return ERROR */
            }

            if( ( bytep = getvardesc( objinfop)) == NULL)
            {       /* formatted an SNMP object identifier object? */
                    univerr = -9;
                    return( ( unsigned long) ERROR);            /* no, return ERROR */
            }

            if( getobjptrs( bytep, &objp) < 0)
            {       /* got obj id pointers? */
                    univerr = -10;
                    return( ( unsigned long) ERROR);            /* no, return ERROR */
}

if( ( objp.endobjp - objp.tagp) > ( buffend - bufferp))
{
                                /* enough room for the obj. identifier? */
            univerr = -11;
           return( ( unsigned long) ERROR);/* no, error */
}

bufferp += movenstr( objp.tagp, bufferp);       /* move the obj. ident into the packet */

objp.tagp = bufferp - ( objp.endobjp - objp.tagp);

/* point to beginning of obj id in the buffer */

if( getobjptrs( objp.tagp, &objp) < 0)
{       /* got obj id ptrs? */

      univerr = -12;
      return( ( unsigned long) ERROR);      /* no, error */
}

if( vblkp->indexlen != 0)
{       /* is there an index to add to the object? */

        if( buffend - bufferp < ( int) vblkp->indexlen)
        {       /* yes, is there room? */

                bufferp = seqbegp;      /* remove the sequence for this var */
        }
        else
        {               /* there is room */
                int i;
                for( i = 0 ; i < ( int) vblkp->indexlen ; i++)
                {       /* move it */
                        *(bufferp++) = vblkp->index[i];


                }
        }
}
else
{
        if( bufferp < buffend)
        *( bufferp++) = '\0';   /* add a zero to the object because its the only instance of the object */
}

if( bufferp >= seqlenp)
{
        if( ( status = setobjlen( ( bufferp - objp.datap), objp.lenp))< 0)
        {
                univerr = -26;
                return( ( unsigned long) ERROR);
```

```
                }

                if( status > 1)
                {       /* object length too long for length field? */

                        univerr = -28;                  /* yes, error */
                        return( ( unsigned long) ERROR);
                }
                if( ( buffend - bufferp) < 2)
                {       /* enough space for a NULL obj id? */

                        univerr = -29;                  /* no, error */
                        return( ( unsigned long) ERROR);
                }

                *( bufferp++) = UNULLID;        /* put in the null type id */
                *( bufferp++) = 0;              /* put in a length of 0 */

                if( ( status = setobjlen( ( bufferp - seqlenp - 1), seqlenp)) < 0)
                {
                        univerr = -30;
                        return( ( unsigned long) ERROR);
                }

                if( status > 1)
                {               /* object length too long for length field? */

                        univerr = -31;                  /* yes, error */
                        return( ( unsigned long) ERROR);
                }


        }

        vblkp = vblkp->nxtblock;
        fprintf( stderr, "getreq: vblkp = %04x\n", vblkp);
}

if( ( status = setobjlen( ( bufferp - varblenp - 1), varblenp)) < 0)
{
        univerr = -32;
        return( ( unsigned long) ERROR);
}

if( status > 1)
{       /* object length too long for length field? */
        univerr = -33;          /* yes, error */
        return( ( unsigned long) ERROR);
}

if( ( status = setobjlen( ( bufferp - reqlenp - 1), reqlenp)) < 0)
{
        univerr = -34;
        return( ( unsigned long) ERROR);
}

if( status > 1)
{       /* object length too long for length field? */
        univerr = -35;          /* yes, error */
        return( ( unsigned long) ERROR);
}

if( ( status = setobjlen( ( bufferp - pdulenp - 1), pdulenp)) < 0)
{       univerr = -36;
        return( ( unsigned long) ERROR);
}

if( status > 1)
{       /* object length too long for length field? */
        univerr = -37;          /* yes, error */
        return( ( unsigned long) ERROR);
```

```
    }

    *buflen = bufferp - buffer;                 /* save the packet length */
    return( *reqidp);                   /* return the request id number */
}

// END GETREQ.C
```

```
/*----- getvardesc() ----------------------------------------*/
/* Given a pointer to a structure defining an MIB variable, format
an SNMP object identifier object for the variable defined by the
structure.  Return a pointer to the byte string of the object
identifier on success, return NULL on error.

*/

#include   <stdio.h>                                    /* for 'fprintf()' */
#include   "\gary\snmp\snmp_src\snmp\snmp.h"            /* for UOBJID, etc. */
#include   "\gary\snmp\snmp_src\mib\mib.h" /* for objinfo struct */

#define    BUFLEN              512

unsigned char *getvardesc( varstructp)
struct objinfo *varstructp;
{
   extern unsigned char isoorgval;
   static unsigned char buffer[BUFLEN];
   unsigned char *bytep;
   struct objinfo *structp;
   int i;
   structp = varstructp;

   bytep = &buffer[BUFLEN - 1]; /* point at the end of the buffer */

   *( bytep--) = '\0';                              /* mark the end of the buffer */

   while( structp->flag > (-16))
   {          /* while we haven't backup up the MIB structure tree to the beginning, loop */

                for( i = 0 ; structp->flag >= 0 ; i++, structp--);
                                    /* back up this sub-structure array until we reach the first structure in the array of structures */
              *( bytep--) = ( unsigned char)i;
                                    /* put the array index of the variable we pointed at in the buffer string */
              if( structp->flag > (-16))              /* are we at the top of the tree? */
                        structp = ( struct objinfo *) structp->ptr;

     /* no, point at the structure in the array of structures that pointed to this array */
   }
   *( bytep--) = isoorgval;

   *bytep = ( unsigned char)( &buffer[BUFLEN - 2] - bytep);

   /* set the length of the object identifier */

   *( --bytep) = UOBJIDID;          /* set the tag of the object identifier */

   return( bytep);          /* return a pointer to the completed object identifier */
}
// END GETVARDE.C
```

```
/*---- searcharr() ----------------------------------------*/
/* Search the MIB for the structure which corresponds to the
variable name pointed to by 'namep'. Start searching the MIB
structure at the point pointed to by 'objinfop'. Return NULL on
error, or a pointer to the appropriate structure on success.

*/

#include   <stdio.h>                                    /* for 'fprintf()', NULL */
#include   "\gary\snmp\snmp_src\mib\mib.h"              /* for objinfo struct */

#define    ERROR              (-1)                      /* error returned status */

struct objinfo *searcharr( objinfop, namep)
    struct objinfo *objinfop;
    char *namep;
{
  int cmpstr( unsigned char *, unsigned char *, int);
  int strlen;
  char *bytep;
  for( strlen = 0, bytep = namep ; *bytep != '\0' ; strlen++, bytep++);

                              /* find the length of the string pointed to by namep */
  strlen++;                   /* make sure the '\0' is counted */

  if( objinfop->flag < 0)
          objinfop++;                              /* skip the zeroth array member */

        /* modified the following line 1-10-91 */

  while( !( ( objinfop->objname == NULL) && ( objinfop->flag == ( -1))))
  {
                              /* loop while the object name pointer in the structure is not NULL and the flag is not -1 */
        if( objinfop->objname != NULL)
        {

              if( cmpstr( ( unsigned char *) objinfop->objname, ( unsigned char *) namep, strlen) == 0)
                                                          /* found an object name matching the namep string? */
                    return( objinfop);    /* yes, return a pointer to the structure */
              else
              {                     /* didn't find a match, check for sub-structures */
                    if( objinfop->flag == 1)
                    {                     /* sub-structures exist? */

                          struct objinfo *substrcp;       /* yes, search them */
                          if( ( substrcp = searcharr( ( struct objinfo *) objinfop->ptr, namep)) != NULL)
                                                          /* found a match in a substructure? */
                                return( substrcp);        /* yes, return the structure pointer */
                    }
              }
        }
        objinfop++;           /* otherwise, point to the next structure */
  }

   return( NULL);    /* didn't find a match, return error */
}
// END SRCHARR.C
```

```
/*----- initbkrefs() ----------------------------------------*/

/* Forward pointer initialization and MIB generation routine.
Attach the name of the MIB files go the MIB stub that comes
with the system.

*/

#include   "\gary\snmp\snmp_src\mib\mib.h"          /* for objinfo struct */
#include   <stdio.h>

#include   <fcntl.h>          /* for open() */
#include   <sys/types.h>              /* for open() */
#include   <sys/stat.h>               /* for open() */
#include   <io.h>                     /* for open(), read(), filelength() */

#include   <stdlib.h>         /* for free(), malloc() */
#include   <malloc.h>                 /* for free(), malloc() */

#include   <conio.h>          /* for getche() */

#include   "\gary\snmp\snmp_src\mib\makemib.h"          /* get mib structures */

#define          UCASEMSK          0x5f     /* mask to make lower case into upper */
#define          ERROR             ( -1)    /* error return status */


/*----- the miblink structure -------------------------------*/
/* Define the 'miblink' structure so a mib can be connected to
the iso-org tree in the correct place.

struct modptrs          {
                        char *ptr[ 10];
                        struct modptrs *nxtblk;
                        } ;

int initbkrefs( char **mibptrs)
{
  extern struct objinfo isoorg[],
                                dod[],
                                internet[],
                                private[],
                                enterprise[],
                                mgmt[];

  extern struct miblink *miblinkroot;

  int adjptrs( struct objinfo *, int, struct strucptr *, struct modptrs *);
  int testmib( struct objinfo *, int);

  struct miblink *mlp, *mlp1;

  struct objinfo *objinfop;
  unsigned char *mib, *bytep;
  char *cp;
  int offset;
  int fd, i;
  long mibsize;

                /* adjust the root mib backward pointers */

  dod[ 0].ptr = ( void *) &( isoorg[ 6]);
  internet[ 0].ptr = ( void *) &( dod[ 1]);
  mgmt[ 0].ptr = ( void *) &( internet[ 2]);
  private[ 0].ptr = ( void *) &( internet[ 4]);
  enterprise[ 0].ptr = ( void *) &( private[ 1]);


        /* now add the new mibs to the root mib */

  if( ( mibptrs == NULL) || ( *mibptrs == NULL))
```

```
{
        fprintf( stderr, "initbkrefs: no mib file given\n");
        return( ERROR);
}

while( *mibptrs != NULL)
{
        cp = *mibptrs;
        for( i = 0; cp[ i] != '\0'; i++)
        {       /* search file name string */

                if( cp[ i] == '\057')           /* is a character a '/'? */
                        cp[ i] = '\134';        /* yes, change it to a '\' */
        }

        if( ( fd = open( cp, ( O_RDONLY | O_BINARY))) < 0)
                fprintf( stderr, "initbkrefs: can't open %s, ignoring it\n", cp);
        else
        {
                if( ( mibsize = filelength( fd)) < 0L)
                {                       /* got file length? */
                        close( fd);
                        fprintf( stderr, "initbkrefs: can't get file length of %s, exiting\n", cp);
                        return( ERROR - 1);
                }
                if( ( mib = ( unsigned char *) malloc( ( size_t) mibsize))== NULL)
                {
                        close( fd);
                        fprintf( stderr, "initbkrefs: can't allocate mib memory, exiting\n");
                        return( ERROR - 2);
                }
                if( ( i = read( fd, ( char *) mib, ( unsigned int) mibsize)) < 0)
                {
                        close( fd);
                        free( ( void *) mib);
                        fprintf( stderr, "initbkrefs: couldn't read the mib from the file\n");
                        return( ERROR - 3);
                }
                if( i != ( int) mibsize)
                {
                        close( fd);
                        free( ( void *) mib);
                        fprintf( stderr, "initbkrefs: mib size was %ld, read %d bytes, exiting\n", mibsize, i);
                        return( ERROR - 4);
                }

                close( fd);

                bytep = mib;            /* point at the mib data segment offset */
                offset = ( int) ( mib - ( unsigned char *) *bytep);

                /* get the mib data segment offset adjustment for the mib in the data segment for this program */

                bytep += sizeof( unsigned char *); /* point at the link data pointer value */

                mlp = ( struct miblink *) ( ( ( unsigned char *) ( *( ( struct miblink **) bytep))) + offset);

                                /* get the location of the link data */
                if( mlp->mibname != NULL)
                {
                        mlp->mibname += offset;
                }
                else
                        fprintf( stderr, "initbkrefs: the mib name pointer is NULL\n");

                if( mlp->rootptr != NULL)
                        mlp->rootptr = ( struct objinfo *) ( ( unsigned char *) mlp->rootptr + offset);
                else
                {
                        fprintf( stderr, "initbkrefs: the pointer to the root of the mib is NULL\n");
                        free( ( void *) mib);
```

```
                return( ERROR - 5);
        }
        if( mlp->listptr != NULL)
                mlp->listptr = ( struct strucptr *) ( ( unsigned char *) mlp->listptr     + offset);
        else
                fprintf( stderr, "initbkrefs:the pointer to the list of structure pointers is NULL\n");
        if( mlp->nextmib != NULL)
        {
                fprintf( stderr, "initbkrefs: the link to the next mib is not NULL\n");
                mlp->nextmib = NULL;
}

if( mlp->rootptr->ptr != NULL)
        fprintf( stderr, "initbkrefs: the backward pointer in the top array is not NULL\n");

if( mlp->rootptr->flag != ( -1))
        fprintf( stderr, "initbkrefs: the flag of the first structure in the array is not -1\n");

if( ( i = adjptrs( mlp->rootptr, offset, mlp->listptr, NULL)) < 0)
{
        fprintf( stderr, "initbkrefs: can't adjust pointers, error %d\n", i);
        free( ( void *) mib);
}
else
{               /* mib pointers were adjusted successfully */
        if( ( i = testmib( mlp->rootptr, 0)) < 0)
        {
                fprintf( stderr, "initbkrefs: MIB %s faulty, error %d\n", mlp->mibname, i);
                free( ( void *) mib);
        }

        else
        {               /* mib seems to be contiguous */
                bytep = mlp->rootentry;

                /* point at the entry point string */
                if( *( bytep++) != 0x2b)
                {       /* is the first byte correct? */

                        fprintf( stderr, "initbkrefs: mib entry point undefined\n");
                        free( ( void *) mib);
                }
                else
                {               /* first mib entry point was defined */
                        int errflag;
                        errflag = 0;
                        objinfop = isoorg;

                    /* find the structure in the iso-org tree where this mib attaches */
                        while( ( *bytep != '\0') && ( errflag < 1))
                        {
                            /* walk the iso-org tree according to the rootentry string */
                                for( i = 0; !( ( objinfop->objname == NULL) &&
                                                ( objinfop->flag == ( -1)) &&
                                                        ( objinfop->ptr == NULL)) &&
                                                        (i < ( int) *bytep); objinfop++, i++);

                              /* did we find a structure to link to? */

                                if( ( objinfop->objname == NULL) &&
                                                ( objinfop->flag == ( -1)) &&
                                                        ( objinfop->ptr == NULL))
                                {
                                        errflag++;              /* nope, error */
                                        fprintf( stderr,"initbkrefs: no structure found to link the mib to\n");
                                }
                                else
                                {
                                        if( ( objinfop->flag == 1) && ( objinfop->ptr != NULL))
                                                objinfop = ( struct objinfo *) objinfop->ptr;
                                        else
```

```
                                        {
                                                if( ( objinfop->flag != 0) && ( objinfop->ptr != NULL))
                                                 errflag++;
                                        }
                                }
                                bytep++;
                        }
                        if( ( errflag < 1) && ( *bytep == '\0'))
                        {
                                        /* found a link structure */
                                        /* link the new mib to the iso-org mib */
                                fprintf( stderr, "initbkrefs: linking the \"%s\" mib at \"%s[%d]\"\n",
                                                        mlp->mibname, ( objinfop - i)->objname, i);
                                objinfop->ptr = ( void *) mlp->rootptr;
                                objinfop->flag = 1;
                                objinfop->objname = mlp->rootptr->objname;
                                mlp->rootptr->ptr = ( void *) objinfop;
                                mlp->rootptr->flag = ( -1);

                            /* are there previously installed mibs? */

                                if( ( mlp1 = miblinkroot) == NULL)

                                        /* no, make this the first */
                                        miblinkroot = mlp;
                                else
                                {               /* there are other mibs */
                                                /* add this one to the end of the chain */
                                        while( mlp1->nextmib != NULL)
                                                mlp1 = mlp1->nextmib;

                                        mlp1->nextmib = mlp;
                                }
                        }
                        else
                        {       /* error */
                              fprintf( stderr, "initbkrefs: counldn't attach the \"%s\" MIB\n", mlp->mibname);

                             fprintf( stderr, "initbkrefs: tried linking the mib at structure %s[%d]\n",
                                                                        ( objinfop - i)->objname, i);
                                free( ( void *) mib);
                        }
                }
        }
    }
    mibptrs++;
  }
  return( 0);
}
```

```
/*----- adjptrs ----------------------------------------------------*/
/* Given a pointer to a array of 'objinfo' structures, adjust all of
the pointers by the offset given. Return zero on success; negative
one on error. When there is a sub-tree of structures, call
'adjptrs' recursively. The end of an array of structures is
indicated by a structure with elements 'objname' and 'ptr' which are
NULL, and 'flag' which is -1.

*/

int adjptrs( struct objinfo *objinfop, int offset, struct strucptr *listptr, struct modptrs *modp)
{
  int addmodptr( char *, struct modptrs *);
  int testptrmod( char *, struct modptrs *);
  int relptrblks( struct modptrs *);
  struct objinfo *findname( char *, struct strucptr *);
  void *malloc( size_t);
  void free( void *);

  struct strucptr *sptr;
  struct objinfo *tstptr;
  int i, relflag;
                                                    /* do this stuff only once */
  if( modp == NULL)
  {
                /* is there a modified pointer list? */
        relflag = 1;            /* no, start one and set a flag */
        if( ( modp = ( struct modptrs *) malloc( ( size_t) sizeof( struct modptrs))) == NULL)
        {
                fprintf( stderr, "adjptrs: memory allocation error\n");
                return( ERROR);
        }       /* initialize the just created block */

        for( i = 0; i < 10; i++)
              modp->ptr[ i] = NULL;

        modp->nxtblk = NULL;

                /* now adjust the pointers in the structure pointer list */
        if( ( sptr = listptr) != NULL)
        {
                  /* adjust the pointers of the structure pointer list */
                while( ( sptr->objname != NULL) || ( sptr->objstruc != NULL))
                {
                        if( sptr->objname != NULL)
                                sptr->objname += offset;
                        else
                                fprintf( stderr, "adjptrs: a pointer list name pointer is null\n");

                        if( sptr->objstruc != NULL)
                                sptr->objstruc = (struct objinfo *)( ( ( char *) sptr->objstruc)+offset);
                        else
                                fprintf( stderr, "adjptrs: a pointer list structure pointer is null\n");
                        sptr++;
                }
        }
}
else                    /* this is not the first iteration of 'adjptrs()' */
        relflag = 0;            /* clear the flag for later use */


while( !( ( objinfop->objname == NULL) && ( objinfop->ptr == NULL) && ( objinfop->flag == ( -1))))
{
        if( objinfop->objname != NULL)
        {       /* is 'objname' not null? */
                objinfop->objname += offset;                    /* yes, adjust it */
                if( ( tstptr = findname( objinfop->objname, listptr))== NULL)

                        /* found object name in object name list? */
                        fprintf( stderr, "adjptrs: %s not in object name list\n", objinfop->objname);
        }
```

```
        if( objinfop->ptr != NULL)
        {       /* is 'ptr' not null? */
                /* yes, adjust it */
                objinfop->ptr = ( struct objinfo *) ( ( unsigned char *) objinfop->ptr + offset);

                        /* is 'ptr' pointing to a sub-array of structures? */

                if( objinfop->flag == 1)
                {
                        if( ( objinfop->objname != NULL) && ( tstptr != ( struct objinfo *) objinfop->ptr))
                        {
                            /* yes, is it pointing to the right structure array? */

                                fprintf( stderr,  "adjptrs: the object pointer for the variable \"%s\" in the variable list",
                                                                                        objinfop->objname);

                                fprintf( stderr,"\n\t doesn't point to the right object\n");
                        }

                        if( adjptrs( ( struct objinfo *) objinfop->ptr, offset,listptr, modp) < 0)
                        {
                            /* can we adjust the pointers of the sub-array? */
                                fprintf( stderr, "adjptrs: can't adjust pointers in the sub-array %s\n",  objinfop->objname);
                        }
                }
                if( objinfop->flag == 0)
                {               /* 'ptr' points to an array of strings? */
                        char **strptr;                                          /* yes, adjust them, too */
                        strptr = ( char **) objinfop->ptr;                      /* point at the first pointer to a string */
                        while( *strptr != NULL)
                        {                       /* while there are pointers */
                                if( testptrmod( ( char *) *strptr, modp) < 0)
                                {
                                    /* is this pointer already modified? */
                                        *strptr += offset;      /* no, adjust it */

                                    /* add it to the list of modified pointers */
                                        if( addmodptr( ( char *) *strptr, modp) < 0)
                                        {
                                                fprintf( stderr, "adjptrs: error on adding a modified pointer\n");
                                                return( ERROR - 1);
                                        }
                                }
                                strptr++;               /* go to the next pointer */
                        }
                        if( ( objinfop->objname != NULL) && ( tstptr != objinfop))
                        {
                            /* does the list pointer point to the right struc? */
                                fprintf( stderr, "adjptrs: object pointer for variable %s in the variable list",
                                                                                        objinfop->objname);
                                fprintf( stderr,"\n\t doesn't point to the right object\n");
                        }
                }
        }

        objinfop++;                     /* point to the next structure in the array */
  }

  if( relflag > 0)
        relptrblks( modp);

  return( 0);
}
```

```
/*----- addmodptr ----------------------------------------------*/
/* Add a pointer to the list of pointers in a MIB that have been
modified and should not be modified again.

*/

int addmodptr( char *ptr, struct modptrs *modblk)
{
   void *malloc( size_t);
   int i;

   for( i = 0; ( i < 10) && ( modblk->ptr[ i] != NULL) && ( modblk->ptr[ i] != ptr); i++);
            if( modblk->ptr[ i] == ptr)                    /* is the pointer already listed? */
                     return( 0);                           /* yes, we're done */
            if( i >= 10)
            {                        /* all 10 pointers tested without a match? */
                     if( modblk->nxtblk != NULL)           /* right, is there another block? */
                              return( addmodptr( ptr, modblk->nxtblk));    /* yes, check it */
                     else
                     {                                     /* there isn't another block, get one */
                              struct modptrs *nxtblkp;

                              if(( nxtblkp = ( struct modptrs *) malloc( ( size_t) sizeof( struct modptrs))) == NULL)
                                       return( ERROR);     /* error on memory allocation */
                                                           /* initialize the new block */
                              nxtblkp->ptr[ 0] = ptr;      /* save the new pointer */

                              for( i = 1; i < 10; i++)     /* initialize the other pointers */
                                       nxtblkp->ptr[ i] = NULL;

                              nxtblkp->nxtblk = NULL;      /* terminate the list of blocks */
                              modblk->nxtblk = nxtblkp;    /* add the new block to the list */
                              return( 0);
                     }
            }                                              /* no match for 'ptr' and found a null pointer */
            modblk->ptr[ i] = ptr; /* add 'ptr' to the list */

     return( 0);
}

/*----- relptrblks ---------------------------------------------*/
/* Release all pointer blocks associated with the given block pointer.

*/

int relptrblks( struct modptrs *blkptr)
{
   void free( void *);

   struct modptrs *nxtblkp;

   while( blkptr != NULL)
   {
            nxtblkp = blkptr->nxtblk;
            free( ( void *) blkptr);
            blkptr = nxtblkp;
   }
   return( 0);
}
```

```
/*---- testptrmod ----------------------------------------------*/
/* Test the given pointer to see if it has already been modified.

*/

int testptrmod( char *ptr, struct modptrs *mptr)
{
  int i;
        /* test this block of pointers for a match */

  for( i = 0; ( i < 10) && ( mptr->ptr[ i] != NULL) && ( mptr->ptr[ i] != ptr); i++);
        if( mptr->ptr[ i] == ptr)           /* found a match? */
              return( 0);                   /* yes, return */

        if( ( i >= 10) && ( mptr->nxtblk != NULL))          /* no match and another block of pointers exists? */
            return( testptrmod( ptr, mptr->nxtblk));  /* yes, test it */

        return( ERROR );                    /* no other block exists, return failure */
}

/*---- testmib -------------------------------------------------*/
/* Test all of the names of the variables, the flag values, and the
type values of the array of mib structures pointed to by the given
pointer.

*/

int testmib( struct objinfo *objinfop, int indent)
{
  int i;

  indent += 2;

  while( !( ( objinfop->objname == NULL) && ( objinfop->ptr == NULL) && ( objinfop->flag == ( -1))))
  {
        if( objinfop->flag == 1)
        {
              if( ( ( struct objinfo *) objinfop->ptr)->ptr != objinfop)
              {
                    fprintf( stderr, "testmib: the backward pointer in the first structure in\n\t\t");
                    fprintf( stderr, "\tthe sub-array \"%s\" does not have the right value\n", objinfop->objname);
                    return( ERROR);
              }
              if( ( ( struct objinfo *) objinfop->ptr)->flag != ( -1))
              {
                    fprintf( stderr, "testmib: the flag in the first structure in the ");
                    fprintf( stderr, "sub-array \"%s\" is not -1\n", objinfop->objname);
                    return( ERROR - 1);
              }
              testmib( objinfop->ptr, indent);
        }
        objinfop++;
  }
  return( 0);
}
```

```
/*---- findname ----------------------------------------*/
/* Find the given object name string in the array of 'structptr'
structures given. Return the object structure pointed to in the
array if the name is found. If the name isn't found, return null.

*/

struct objinfo *findname( char *name, struct strucptr *arrayp)
{
  int i;
  char *cp;

  while( ( arrayp->objname != NULL) || ( arrayp->objstruc != NULL))
  {
        cp = arrayp->objname;
        for( i = 0; ( cp[ i] != '\0') && ( cp[ i] == name[ i]); i++);
        if( cp[ i] == name[ i])
              return( arrayp->objstruc);
        arrayp++;
  }
  return( NULL);
}

// END INITBKRF.C

/*---- initmib() ---------------------------------------*/
/* Initialize the Management Information Base (MIB) for use in
analyzing Simple Network Management Protocol (SNMP) protocol data
units (PDUs).

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"                    /* for struct routinfo */
#include   "\gary\snmp\snmp_src\mib\mib.h"                  /* for struct objinfo */

#include   <stdio.h>  /* define NULL */

#define    ERROR              (-1)

int initmib( char *dbfilename, char **mibptr)
{
  extern struct routinfo *firstrinfop;
  extern int univerr;
  extern struct objinfo *isoorgroot;

  int initbkrefs( char **);
  struct routinfo *analdb( char *);
  int ifIndex_f( char *);
  struct objinfo *searcharr( struct objinfo *, char *);

  struct routinfo *rinfop;
  struct objinfo *objinfop;
  int i;

  if( ( i = initbkrefs( mibptr)) != 0)
  {       /* initialize the MIB references */

        fprintf( stderr, "initmib: error %d initializing the mib\n", i);
        return( ERROR);
  }

  if( ( objinfop = searcharr( isoorgroot, "ifIndex")) == NULL)
  {
        fprintf( stderr, "initmib: couldn't find ifIndex in the mib\n");
        fprintf( stderr, "\ttrap reporting may not work correctly\n");
        fprintf( stderr, "\ta standard MIB must be included for this to work\n");
  }

  objinfop->funct = ifIndex_f;    /* used by trap analyzer routine */

  if( ( firstrinfop = analdb( dbfilename)) == NULL)
```

```
    {           /* fatal error analyzing the router database? */

                fprintf( stderr, "initmib: analdb error %d\n", univerr);
                return( ERROR - 2);                     /* yes, error */
    }

    if( univerr != 0)
    {                       /* any non-fatal errors? */

                if( univerr == -6)    /* yes, bad format error? */
                fprintf( stderr, "initmib: possible database format error\n");
    }

    return( 0);
}

// END INITMIB.C

/*---- mib 2 ------------------------------------------*/
/* The MIB II definition of the Management Information Base for
network devices.

*/

/*---- mibstruc.c -------------------------------------------*/
#include    <stdio.h>                                         /* for NULL */
#include    "\gary\snmp\snmp_src\mib\mib.h"             /* for 'struct objinfo' def */
/* #include "\gary\snmp\snmp_src\snmp\snmp.h"           /* for IpAddrID, etc. */
#include    "\gary\snmp\snmp_src\mib\makemib.h"                 /* for 'struct miblink' def */

/*---- name strings -----------------------------------------*/

char enterprise_s[] = "enterprise";

struct objinfo enterprise[] = {
            { enterprise_s, -1, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
            { NULL, 0, NULL, -1, NULL},
```

```
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, 0, NULL, -1, NULL},
                { NULL, -1, NULL, -1, NULL} };

char private_s[] = "private";

struct objinfo private[] = {
                            { private_s, -1, NULL, -1, NULL},
                            { enterprise_s, 1, ( void *) enterprise, -1, NULL},
                            { NULL, -1, NULL, -1, NULL} };

char mgmt_s[] = "mgmt";

struct objinfo mgmt[] = {
                           { mgmt_s, -1, NULL, -1, NULL},
                           { NULL, 0, NULL, -1, NULL},
                           { NULL, -1, NULL, -1, NULL} };

char internet_s[] = "internet";

struct objinfo internet[] = {
                             { internet_s, -1, NULL, -1, NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { mgmt_s, 1, ( void *) mgmt, -1, NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { private_s, 1, ( void *) private, -1, NULL},
                             { NULL, -1, NULL, -1, NULL} };

char dod_s[] = "dod";

struct objinfo dod[] = {
                          { dod_s, -1, NULL, -1, NULL},
                          { internet_s, 1, ( void *) internet, -1, NULL},
                          { NULL, -1, NULL, -1, NULL} };

char isoorg_s[] = "iso org";

struct objinfo isoorg[] = {
                             { isoorg_s, -16, NULL, -1 , NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { NULL, 0, NULL, -1, NULL},
                             { dod_s, 1, ( void *) dod, -1, NULL},
                             { NULL, -16, NULL, -1 , NULL} };

/*----- base values ------------------------------------*/

unsigned char mibvalue[] = { 0x2b, 0 };
char mibnm[] = "iso org dod internet mgmt";
struct objinfo *mibroot = isoorg;
unsigned char isoorgval = 0x2b;
struct objinfo *isoorgroot = isoorg;
struct miblink *miblinkroot = NULL;
```

```
/*----- structure pointers ------------------------------------*/


struct objinfo *dod_strc = dod;
struct objinfo *internet_strc = internet;
struct objinfo *private_strc = private;
struct objinfo *enterprise_strc = enterprise;
struct objinfo *mgmt_strc = mgmt;


// END MIBSTRUC.C

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include      "\gary\snmp\snmp_src\screen\snmpmgr.h"

void set_maindisp  (void);
void init_clock_win (void);
void init_main_win  (void);

// ----------cleanexit ()----------------------------------------
//

FUNCTION void cleanexit (void)
{
            // Cleanup Time: do your thing here

            textmode(C80);
            textbackground(BLACK);
            textcolor(LIGHTGRAY);
            window(1,1,80,25);
            clrscr();

            return;

}
// ----------------------set_maindisp ()---------------------------------
//

void set_maindisp(void)
{
            name_win();
            info_win();
            init_clock_win();
            init_main_win();
}
```

```
// ----------------------init_win ()-----------------------------------
//

void init_win(void)
{
        struct text_info init_info;

        textmode(C4350);
        clrscr();
        gettextinfo(&init_info);

        Win_clk_left   = (int) init_info.winright - 11;
        Win_clk_top    = (int) init_info.winbottom - 3;
        Win_clk_right  = (int) init_info.winright;
        Win_clk_bottom = (int) init_info.winbottom;

        Win_info_left  = (int) init_info.winleft;
        Win_info_top   = (int) Win_clk_top;
        Win_info_right = (int) Win_clk_left - 2;
        Win_info_bottom = (int) Win_clk_bottom;

        Win_name_left  = (int) init_info.winleft;
        Win_name_top   = (int) init_info.wintop;
        Win_name_right = (int) init_info.winright;
        Win_name_bottom = (int) init_info.wintop + 2;

        Win_main_left  = (int) init_info.winleft;
        Win_main_top   = (int) Win_name_bottom + 2;
        Win_main_right = (int) init_info.winright;
        Win_main_bottom = (int) Win_clk_top - 2;

}
// ---------------clock_win() ----------------------------------------
//

void clock_win(void)
{
        textbackground(CYAN);
        textcolor(YELLOW);
        window(Win_clk_left, Win_clk_top, Win_clk_right, Win_clk_bottom);

}
// ------------init_clock_win ()-----------------------------------------------
//

void init_clock_win(void)
{
        clock_win();
        clrscr();

}
// ----------------------------name_win ()-----------------------------
//

void name_win(void)
{
        textbackground(CYAN);
        textcolor(BLUE);
        window(Win_name_left, Win_name_top, Win_name_right, Win_name_bottom);
        clrscr();
        gotoxy(2,2);
        cprintf("%-26s %s %26s",PROGNAME,TITLE,PGM_VERSION);
}
```

```
// ------------------info_win ()-----------------------------------
//

void info_win(void)
{
        int len=15, sp=2;
        int f1_l= Win_info_left, f1_t= Win_info_top, f1_r=f1_l+len, f1_b=Win_info_bottom;
        int f2_l= f1_l+len+sp, f2_t= Win_info_top, f2_r=f1_l+(2*len)+sp, f2_b=Win_info_bottom;
        int f3_l= f2_l+len+sp, f3_t= Win_info_top, f3_r=f2_l+(2*len)+sp, f3_b=Win_info_bottom;
        int f4_l= f3_l+len+sp, f4_t= Win_info_top, f4_r=f3_l+(2*len)+sp, f4_b=Win_info_bottom;

        textbackground(CYAN);

        window(f1_l, f1_t, f1_r, f1_b);
        clrscr();
        textcolor(RED);
        gotoxy(8,2);
        cprintf("F1");
        textcolor(BROWN);
        gotoxy(5,4);
        cprintf("Show All");

        window(f2_l, f2_t, f2_r, f2_b);
        clrscr();
        textcolor(RED);
        gotoxy(8,2);
        cprintf("F2");
        textcolor(BROWN);
        gotoxy(4,4);
        cprintf("Debug Level");

        window(f3_l, f3_t, f3_r, f3_b);
        clrscr();
        textcolor(RED);
        gotoxy(8,2);
        cprintf("F3");
        textcolor(BROWN);
        gotoxy(3,4);
        cprintf("Ack. Warning");

        window(f4_l, f4_t, f4_r, f4_b);
        clrscr();
        textcolor(RED);
        gotoxy(4,2);
        cprintf("<CTL><END>");
        textcolor(BROWN);
        gotoxy(7,4);
        cprintf("Exit");
}
// ------------main_win ()-----------------------------------------
//

void main_win(void)
{
        textbackground(WHITE);
        textcolor(BLACK);
        window(Win_main_left, Win_main_top, Win_main_right, Win_main_bottom);
}
// ------------init_main_win ()-----------------------------------
//

void init_main_win(void)
{
        main_win();
        clrscr();
        gotoxy(1,2);

}

// end display.c
```

```
//------------------keys.c-----------------------------------

//    Process each keystroke, converting key sequences (such
//    as function keystrokes) into a single value.


// includes

#include "\GARY\SNMP\SNMP_SRC\SCREEN\SNMPMGR.h"


// local function prototypes

// -------------getkey ()-----------------------------------------

FUNCTION int getkey (void)
{
        int key;


        /* loop till a keystroke appears. While we are waiting, update
         * the time and display
         */

        while (bioskey (1) == 0)
        {
                get_time_now ();
                disp_time ();
        }

        key = getch ();

        /* was it a multikey keystroke? If so, get the remainder of the
         * sequence. Translate to canonical value
         */

        if (key == 0)
        {
                key = getch () | FKEYBIT;
        }

        return key;
}

// ---------------async_getkey ()-----------------------------------------

FUNCTION int async_getkey (void)
{
        int key;


        // check for a keystroke. Do not wait if there are none available.

        if (bioskey (1) == 0)
        {
                return 0;
        }

        key = getch ();

        /* was it a multikey keystroke? If so, get the remainder of the
         * sequence. Translate to canonical value
         */

        if (key == 0)
        {
                key = getch () | FKEYBIT;
        }

        return key;
}
```

```
// ----------------key_main ()-----------------------------------

FUNCTION void key_main (void)
{
        int key, i;
        extern int verbose;
        char tmpbuf[ 16];

        key = async_getkey ();

        switch (key)
        {
                case 0:      // no keypress present, simply return
                             break;

                case F1:
                             showall();
                             break;

                case F2:

                             cprintf( "\r\nenter debug level (0-2) ");
                             for( i = 0; ( ( char)( tmpbuf[ i] =
                                          ( char) getche()) != '\r') && ( i < 15); i++);
                             if( tmpbuf[ 0] != '\r')
                             {
                               tmpbuf[ i] = '\0';
                               verbose = atoi( tmpbuf);
                             }
                             cprintf( "debug level set to %d    \n\r\n",verbose);
                             break;

                case F3:
                             warnenable = 0;
                             break;

                case CTRL_END:
                             G_pgmrun = FALSE;
                             break;

                default:
                             beep ();
                             break;
        }

        return;
}

// end keys.c

//----------------------TIME.C-----------------------------
// handel the clock on the screen

// Include files

#include "\GARY\SNMP\SNMP_SRC\SCREEN\SNMPMGR.h"

// local variables

static int old_second;

static int current_second;
static int current_minute;
static int current_hour;
static int current_monthday;
static int current_month;
static int current_year;
static int current_weekday;
static int current_julianday;
```

```
static int prev_second;
static int prev_monthday;

static char Timebuf [10];
static char Datebuf [10];
static char Delaybuf[10];

static int DisplayTime = FALSE;


// ------------------get_time_now ()-----------------------------------
// This function gets the current time, including julian date.

FUNCTION void get_time_now (void)
{
        struct tm *timeptr;
        time_t secsnow;


        old_second = current_second;   // remember the 'old' second value

        (void) time ( &secsnow );
        timeptr = localtime (&secsnow);

        current_second   = timeptr -> tm_sec;
        current_minute   = timeptr -> tm_min;
        current_hour     = timeptr -> tm_hour;
        current_monthday = timeptr -> tm_mday;
        current_month    = timeptr -> tm_mon + 1;
        current_year     = timeptr -> tm_year;
        current_weekday  = timeptr -> tm_wday;
        current_julianday = timeptr -> tm_yday + 1;

        sprintf (Timebuf, "%02d:%02d:%02d", current_hour,
                                                current_minute,
                                                current_second);

        sprintf (Datebuf, "%02d/%02d/%02d", current_month,
                                                current_monthday,
                                                current_year );

        return;
}  // end get_time_now()
```

```
// ------------------disp_time ()-----------------------------------
// This function updates the time and date on the screen.

FUNCTION void disp_time (void)
{
        int savex, savey;


        if (current_second != prev_second)
        {
                prev_second = current_second;

                savex = wherex();
                savey = wherey();

                clock_win();
                gotoxy(3,2);
                cprintf(Timebuf);


                if (current_monthday != prev_monthday)
                {
                        prev_monthday = current_monthday;
                        gotoxy(3,3);
                        cprintf(Datebuf);
                }

                main_win();
                gotoxy(savex, savey);

        }
}

// ---------------get_timestring ()-----------------------------------
// this function returns a pointer to the timestring. */

FUNCTION const char * get_timestring (void)
{
        return Timebuf;
}

// ---------------get_datstring ()-----------------------------------
// this function returns a pointer to the datestring. */

FUNCTION const char * get_datestring (void)
{
        return Datebuf;
}


// end time.c

// misc.cp - misc routines

#include "\gary\snmp\snmp_src\screen\snmpmgr.h"

// ---------------beep ()-----------------------------------
// beep - make a cute sound

FUNCTION void beep (void)
{
  sound (880);
  delay (300);   // 300 msec
  nosound ();

  return;
}
// end misc.c
```

Appendix B

```
/*----- installpd() ----------------------------------------*/
/* Install the packet driver at the first unused software interrupt between
060h and 080h, inclusive. Installation consists of setting the software
interrupt vector to point to the dummy packet driver 'dummypd'. This
routine is useful only with the dummy packet driver.

*/

#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                /* for int86x() */

#define    ERROR                  ( -1)

int installpd( void)
{
           void interrupt far dummypd( void);
           void ( interrupt far *_dos_getvect( unsigned int))();
           void _dos_setvect( unsigned int, void (interrupt far *)());
           unsigned int vec = 0x60;
           void (interrupt far *intp)();
           int status;

           do
           {          /* loop while we haven't found a null vector and there are
                                                    still vectors to search */
                      intp = _dos_getvect( vec);    /* get the vector pointer */
                      vec++;                        /* point to the next vector */

           } while( ( intp != NULL) && ( vec < ( ( unsigned int) 0x81)));

           if( intp != NULL)
           {                    /* found a null vector? */

                     return( ERROR);                /* no, error */
           }

           vec--;                           /* get the null vector pointer */
           _dos_setvect( vec, dummypd);     /* point the null vector at the dummy packet driver */
           return( ( int) vec);             /* return the vector of the dummy packet driver */
}

// END INSTLPD.C
```

```
/*----- prdvrerr() ----------------------------------------*/
/*  Given a driver error number, return a pointer to the appropriate
error message.

*/

#include   <stdio.h>                /* for NULL */

char errmsg1[] = "bad handle",
             errmsg2[] = "wrong interface class",
             errmsg3[] = "wrong interface type",
             errmsg4[] = "wrong interface number",
             errmsg5[] = "bad packet type",
             errmsg6[] = "multicast not supported",
             errmsg7[] = "driver can't terminate",
             errmsg8[] = "mode not supported",
             errmsg9[] = "insufficient space",
            errmsg10[] = "type in use",
            errmsg11[] = "unsupported/bad command",
            errmsg12[] = "can't send the packet",
            errmsg13[] = "can't set hardware address",
            errmsg14[] = "bad hardware address/length";


char *errmsgp[] = { NULL, errmsg1, errmsg2, errmsg3, errmsg4, errmsg5,
                        errmsg6, errmsg7, errmsg8, errmsg9, errmsg10,
                        errmsg11, errmsg12, errmsg13, errmsg14, NULL
                    };

char *prdvrerr( int errno)
{
            extern char *errmsgp[];
            int i;
            for( i = 1; ( i < errno) && ( errmsgp[ i] != NULL); i++);
            return( errmsgp[ i]);
}


// END PRDRERR.C
```

```
/*---- relbuffer() ----------------------------------------*/
/*  Release the packet buffer beginning at 'buffer' and the buffer block
in the chain of buffer blocks that points to 'buffer'.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"  /* get packet driver header */
#include   <stdio.h>                                   /* for fprintf(), etc. */
#include   <dos.h>                                     /* for int86x() */

int relbuffer( struct bufferblk far *buffer)
{
        extern struct bufferblk * far bufroot;
        int far relbuf( unsigned char far *);
        struct bufferblk far *bufblkp, far *prevblkp;
        unsigned char far *bufp;
        int status;

        if( bufroot == NULL)
        {       /* are there any buffer blocks? */
                fprintf( stderr, "relbuffer: bufroot is NULL\n");

                buffer->usedflag = 0;  /* no, clear this buffer anyway */
                return( 0);
        }

        bufp = ( unsigned char far *) &bufroot;

        if( FP_SEG( buffer) != FP_SEG( bufp))
                return( ERROR);

        prevblkp = bufblkp = ( struct bufferblk far *) bufp;
        FP_OFF( bufblkp) = ( unsigned int) bufroot;
        FP_OFF( prevblkp) = NULL;       /* initialize the previous block pointer */

        while( ( bufblkp->nxtblockp != NULL) && ( bufblkp != buffer))
        {
                /* while there are blocks and we haven't found the right one, loop */
                prevblkp = bufblkp;                     /* point to the current block */
                FP_OFF( bufblkp) = ( unsigned int) bufblkp->nxtblockp;     /* point to the next block */
        }

        if( bufblkp != buffer)
        {
                        /* right block found? */

                fprintf( stderr, "relbuffer: buffer at %04x:%04x not on chain\n", FP_SEG( buffer), FP_OFF( buffer));
                fprintf( stderr, "\treleasing it anyway\n");

                buffer->usedflag = 0;                   /* no, clear this buffer anyway */
                return( 0);
        }

        bufblkp->usedflag = 0;

        if( FP_OFF( prevblkp) == NULL)
        {                       /* are we trying to release the first block? */
                bufroot = bufblkp->nxtblockp;     /* yes, change the root pointer */
        }
        else                                    /* not trying to release the first block */
                prevblkp->nxtblockp = bufblkp->nxtblockp;  /* point to the following block from the previous one */
        return( 0);
}

// END RELBUFFR.C
```

```
/*----- relhandle() ----------------------------------------*/
/*  Release the packet driver from receiving packets associated with
the given 'handle'.

*/

#include    <stdio.h>               /* for fprintf(), etc. */
#include    <dos.h>                         /* for int86x() */

#define     ERROR               ( -1)

int relhandle( int handle)
{
        extern int pdvector;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        char *prdvrerr( int);
        struct SREGS segregs;
        union REGS regs;

        segread( &segregs);             /* get the segment registers */
        regs.h.ah = 3;                  /* set release_type() function number */
        regs.x.bx = ( unsigned int) handle;         /* set the handle */
        int86x( pdvector, &regs, &regs, &segregs);  /* call the software int. */
        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "relhandle: error on int86x call\n");
                fprintf( stderr, "\tdriver error was %s\n", prdvrerr( ( int) regs.h.dh));

                return( ERROR);                                 /* yes, error */
        }

        return( 0);
}

// END RELHANDL.C
```

```
/*----- resetintf() ------------------------------------------------*/
/*  Given a handle, reinitialize the interface associated with it.

*/

#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                        /* for int86x() */

#define    ERROR                  ( -1)

int resetintf( int handle)
{
        extern int pdvector;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        char *prdvrerr( int);
        struct SREGS segregs;
        union REGS regs;

        fprintf( stderr, "resetintf:\n");

        segread( &segregs);                                     /* get the segment registers */
        regs.h.ah = 7;                                          /* set reset_interface() function number */
        regs.x.bx = ( unsigned int) handle;             /* set the handle */
        int86x( pdvector, &regs, &regs, &segregs);              /* call the software int. */

        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "resetintf: error on int86x call\n");
                fprintf( stderr, "\tdriver error was %s\n", prdvrerr( ( int) regs.h.dh));

                return( ERROR);                                 /* yes, error */
        }

        return( 0);

}

// END RESERINF.C
```

```
/*----- sendpkt() ------------------------------------------------*/
/* Send a packet in the buffer 'buffer'.

*/

#include   <stdio.h>              /* for fprintf(), etc. */
#include   <dos.h>                        /* for int86x() */

#define    ERROR                  ( -1)

int sendpkt( unsigned char *buffer, int bufflen)
{
        extern int pdvector;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        char *prdvrerr( int);
        struct SREGS segregs;
        union REGS regs;

//  fprintf( stderr, "sendpkt:\n");

        segread( &segregs);                          /* get the segment registers */
        regs.h.ah = 4;                               /* set send_pkt() function number */
        regs.x.cx = ( unsigned int) bufflen;/* set the buffer length */
        regs.x.si = ( unsigned int) buffer;          /* set the buffer address */
        int86x( pdvector, &regs, &regs, &segregs);   /* call the software int. */

        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "sendpkt: error on int86x call\n");
                fprintf( stderr, "\tdriver error was %s\n", prdvrerr( ( int) regs.h.dh));

                return( ERROR);                      /* yes, error */
        }

        return( 0);
}

// END SENDPKT.C

/*----- showchain() ----------------------------------------------*/
/* Show information about the chain of buffer blocks pointed to by
'bufroot'.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet driver header */
#include   <stdio.h>                                        /* for fprintf(), etc. */
#include   <dos.h>                                          /* for int86x() */

void showchain( void)
{
        extern struct bufferblk * far bufroot;
        struct bufferblk far *bufblkp, far *tmpblkp;
        unsigned char far *bufp;
        int i;

        if( bufroot == NULL)
                fprintf( stdout, "there is no chain of buffers\n");
        else
        {
                bufp = ( unsigned char far *) &bufroot;
                bufblkp = ( struct bufferblk far *) bufp;
                FP_OFF( bufblkp) = ( unsigned int) bufroot;

                while( FP_OFF( bufblkp) != NULL)
                {
                        for( i = 0; i < ( sizeof( struct bufferblk) - E_MX_PSIZ); i++)
                        {
                                if( ( i % 25) == 0)
```

```
                                        putc( '\n', stderr);
                                        fprintf( stderr, "%02x ", ( ( unsigned char far *) bufblkp)[i]);
                                }

                                putc( '\n', stderr);
                                fprintf( stdout, "the buffer block is at %04x:%04x\n", FP_SEG( bufblkp), FP_OFF( bufblkp));

                                fprintf( stdout, "\tthe next block is at %04x:%04x\n", FP_SEG( bufblkp), bufblkp->nxtblockp);

                                fprintf( stdout,"\tthe handle is %d, the buffer length is %d\n", bufblkp->handle, bufblkp->bufferlen);

                                fprintf( stdout, "\tthe buffer is at %04x:%04x\n", FP_SEG( bufblkp), bufblkp->data);

                                FP_OFF( bufblkp) = ( unsigned int) bufblkp->nxtblockp;
                        }
                }
        }

// END SHWCHAIN.C

/*---- terminate()-----------------------------------------*/
/* Given the handle of a type of packet being accepted by the packet
driver, tell the driver to stop receiving that type of packet, to
cease functioning, and to release the memory used by the driver.

*/

#include    <stdio.h>           /* for fprintf(), etc. */
#include    <dos.h>                     /* for int86x() */

#define     ERROR               ( -1)

int terminate( int handle)
{
        extern int pdvector;
        void segread( struct SREGS *);
        int int86x( int, union REGS *, union REGS *, struct SREGS *);
        char *prdvrerr( int);
        struct SREGS segregs;
        union REGS regs;

        fprintf( stderr, "terminate:\n");

        segread( &segregs);                     /* get the segment registers */
        regs.h.ah = 5;                          /* set terminate() function number */
        regs.x.bx = ( unsigned int) handle; /* set the handle */
        int86x( pdvector, &regs, &regs, &segregs);  /* call the software int. */

        if( regs.x.cflag != 0)
        {                       /* error on software interrupt? */

                fprintf( stderr, "terminate: error on int86x call\n");
                fprintf( stderr, "\tdriver error was %s\n", prdvrerr( ( int) regs.h.dh));

                return( ERROR);                                 /* yes, error */
        }

        return( 0);
}

// END TERMIN.C



/*---- addarpent() ----------------------------------------*/
/* Given a far pointer to an ARP reply packet, add the necessary data
to the ARP table.

*/
```

```
#include   <malloc.h>                        /* for malloc(), etc. */
#include   <stdio.h>              /* for NULL */
#include   "\gary\snmp\snmp_src\pd\pdnet.h"

int addarpent( struct arppkt far *arppktp)
{
        extern struct arpentry *arptblqptr;
        void *malloc( size_t);
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        int fmovestr( unsigned char far *, unsigned char far *, int);
        struct arpentry *arpentp;
        int status;


        if( arptblqptr != NULL)
        {
                int endflag;
                endflag = 0;
                arpentp = arptblqptr;
                do
                {
                        status = fcmpstr( ( unsigned char far *) arpentp->ipaddr,
                        ( unsigned char far *) arppktp->locprotoad, IP_ADD_SIZ);

                        if( ( arpentp->nxtentryp == NULL) || ( status == 0))
                                endflag++;
                        else
                                arpentp = arpentp->nxtentryp;

                } while( endflag == 0);

                if( status == 0)
                {

                        fmovestr( ( unsigned char far *) arppktp->locethad,
                                                ( unsigned char far *) arpentp->ethaddr, E_ADD_SIZ);
                        return( 0);
                }

                if( ( arpentp->nxtentryp =
                        ( struct arpentry *) malloc( sizeof( struct arpentry))) == NULL)
                {
                        fprintf( stderr, "addarpent: can't get memory\n");
                        return( ERROR);
                }

                arpentp = arpentp->nxtentryp;
        }

        else
        {
                if( ( arptblqptr = arpentp = ( struct arpentry *) malloc( sizeof( struct arpentry)))== NULL)
                {

                        fprintf( stderr, "addarpent: can't get memory\n");
                        return( ERROR);
                }
        }

        arpentp->nxtentryp = NULL;

        fmovestr( ( unsigned char far *) arppktp->locethad, ( unsigned char far *) arpentp->ethaddr, E_ADD_SIZ);

        fmovestr( ( unsigned char far *) arppktp->locprotoad, ( unsigned char far *) arpentp->ipaddr, IP_ADD_SIZ);

        return( 0);
}
// END ADDARPEN.C
```

```
/*---- arphdlr() ----------------------------------------------*/
/* Respond to ARP packets taken from the packet queue.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <dos.h>                    /* for FP_OFF(), etc. */
#include   <stdio.h>  /* for fprintf(), etc. */

int arphdlr( struct bufferblk far *bufferblkp)
{
        extern struct dvrinfo pdvrinfo;
        int sendarprep( struct arppkt far *);
        int addarpent( struct arppkt far *);
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        extern struct arpentry *arptblqptr;
        struct ethhdr far *ethhdrp;
        struct arppkt far *arppktp;
        int status;
        ethhdrp = ( struct ethhdr far *) bufferblkp->data;
        if( ethhdrp->type != BYTESWAP( E_TYPE_ARP))
        {       /* is it an arp pkt? */

                fprintf( stderr, "arphdlr: got non-ARP packet\n");

                bufferblkp->usedflag = 0;          /* no, release the buffer */
                return( ERROR);
        }

        arppktp = ( struct arppkt far *) ( ethhdrp + 1);
        if( fcmpstr( ( unsigned char far *) arppktp->remprotoad,  ( unsigned char far *) pdvrinfo.locipaddr, IP_ADD_SIZ) != 0)
        {
                        /* is the arp packet for us? */
                bufferblkp->usedflag = 0;          /* no, release the buffer */
                return( 0);
        }

        if( arppktp->operation == BYTESWAP( ARP_OP_REQ))
        {       /* is it an arp req? */

                status = sendarprep( arppktp);

                if( status < 0)
                        fprintf( stderr, "arphdlr: error %d on sendarprep()\n", status);

                bufferblkp->usedflag = 0;          /* release the buffer */
                return( status);
        }
        else
        {
                if( arppktp->operation == BYTESWAP( ARP_OP_REP))
                {       /* is it an arp reply? */
                        status = addarpent( arppktp);
                        if( status < 0)
                                fprintf( stderr, "arphdlr: error %d on addarpent()\n", status);
                        bufferblkp->usedflag = 0;
                        return( status);
                }
                else
                {
                        fprintf( stderr, "arphdlr: received something not a request nor a reply\n");
                        bufferblkp->usedflag = 0;                       /* release the buffer */
                        return( 0);
                }
        }
}

// END ARPHDLR.C
/*---- getethaddr() -------------------------------------------*/
/* Given a pointer to a protocol (Internet) address of a host,
return a pointer to the hardware address of that host.  Check if
```

```
the host is on the local protocol subnet.  If it is, return the
pointer to the host's Ethernet address.  If it isn't, return the
pointer to the local gateway's Ethernet address.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <malloc.h>                    /* for malloc(), etc. */
#include   <stdio.h>                     /* for NULL */

unsigned char *getethaddr( unsigned char *host)
{
        extern struct arpentry *arptblqptr;
        extern int arphandle;
        extern struct dvrinfo pdvrinfo;
        void far buffmgr( void);
        void *malloc( size_t);
        void free( void *);
        char *prdvrerr( int);
        int wait( unsigned int);
        int initdvr( struct dvrinfo *, unsigned char *, int, char far *);
        unsigned char *srcharptbl( unsigned char *);
        int movestr( unsigned char *, unsigned char *, int);
        int sendpkt( unsigned char *buffer, int bufflen);
        int getpkts( void);
        unsigned char *arpaddrp, *host1;
        struct arppkt *arppktp;
        struct ethhdr *ethhdrp;
        int status, retrycnt, i;
        unsigned char broadaddr[ E_ADD_SIZ];
        unsigned char nulladdr[ E_ADD_SIZ];

        if( ( arpaddrp = srcharptbl( host)) != NULL)
        {       /* is there an entry in the arp table for this host? */
                return( arpaddrp);              /* yes, return the pointer */
        }                                       /* otherwise, send an arp request packet */

        for( i = 0; ( i < IP_ADD_SIZ) && ( pdvrinfo.locipmask[ i] != 0) &&
                        ( ( pdvrinfo.locipaddr[ i] & pdvrinfo.locipmask[ i]) ==
                                ( host[ i] & pdvrinfo.locipmask[ i])); i++);
                if( i == IP_ADD_SIZ)                    /* host and local ip addresses the same? */
                        return( pdvrinfo.locethaddr);   /* yes, return a pointer to the local ethernet address */
                if( pdvrinfo.locipmask[ i] != 0)
                {       /* is the host on the local network? */

                        if( ( arpaddrp = srcharptbl( pdvrinfo.locgw)) == NULL)
                        {       /* no, do we have the gateway address instead? */

                                host1 = pdvrinfo.locgw;         /* no, go arp for it */
                        }
                        else
                        {               /* we have the gateway's ethernet address, make
                                           an arp entry for the host with the gateway's
                                           ethernet address in it */

                                struct arpentry *arpentryp;
                                arpentryp = arptblqptr;
                                while( arpentryp->nxtentryp != NULL) /* find the end of the queue */
                                arpentryp = arpentryp->nxtentryp;
                                if( ( arpentryp->nxtentryp = ( struct arpentry *) malloc( sizeof( struct arpentry)))
                                                                                        == NULL)
                                {       /* got memory for an entry? */
                                        fprintf( stderr, "getethaddr: can't get memory for an arp entry\n");
                                        return( arpaddrp);              /* return the gateway's address anyway */
                                }                                       /* initialize the arp table entry */

                                arpentryp = arpentryp->nxtentryp; /* point to the new entry */
                                arpentryp->nxtentryp = NULL;            /* terminate the queue */
                                movestr( host, arpentryp->ipaddr, IP_ADD_SIZ);
                                                        /* move the host's ip address into the entry */
```

```
                    movestr( arpaddrp, arpentryp->ethaddr, E_ADD_SIZ);
                                        /* move the gateway's ethernet address into the entry */
                    return( arpentryp->ethaddr);        /* point to the ethernet addr */
            }
    }
    else                    /* the host is on the local network, get its address */
            host1 = host;

    /* initialize broadcast and null ethernet address strings */

    for( status = 0; status < E_ADD_SIZ; status++)
    {
            broadaddr[ status] = 0xff;
            nulladdr[ status] = '\0';
    }

    if( arphandle < 0)
    {       /* is an arp handle active? */
            int arptype;            /* no, activate one */
            arptype = BYTESWAP( E_TYPE_ARP);
            if( ( arphandle = initdvr( &pdvrinfo, ( unsigned char *) &arptype,
                                                    sizeof( arptype), ( char far *) buffmgr)) < 0)
            {

                    fprintf( stderr, "getethaddr: can't open arp type, error %s\n", prdvrerr( arphandle));
                    return( NULL);
            }
    }

    /* send out an arp request packet */

    if( ( ethhdrp = ( struct ethhdr *) malloc( ( size_t) E_MX_PSIZ)) == NULL)
    {
            fprintf( stderr, "getethaddr: can't get a packet buffer\n");
            return( NULL);
    }

    /* initialize the arp request packet */

    movestr( pdvrinfo.locethaddr, ethhdrp->srcaddr, E_ADD_SIZ);     /* set the ethernet source address */

    movestr( broadaddr, ethhdrp->destaddr, E_ADD_SIZ);              /* set the ethernet destination address */

    ethhdrp->type = BYTESWAP( E_TYPE_ARP);                          /* set the enet type field */

    arppktp = ( struct arppkt *) ( ethhdrp + 1);                    /* point at the arp pkt */

    arppktp->hdwrtype = BYTESWAP( ARP_HDWR_ETH);                    /* set the hdwr type */

    arppktp->prototype = BYTESWAP( E_TYPE_IP);                      /* set the protocol type */

    arppktp->hdwradlen = ( unsigned char) E_ADD_SIZ;                /* set the hdwr address field length */

    arppktp->protoadlen = ( unsigned char) IP_ADD_SIZ;              /* set the protocol address field length */

    arppktp->operation = BYTESWAP( ARP_OP_REQ);                     /* set the arp type to request */

    movestr( pdvrinfo.locethaddr, arppktp->locethad, E_ADD_SIZ);/*set the local ethernet address in the arp pkt */

    movestr( pdvrinfo.locipaddr, arppktp->locprotoad, IP_ADD_SIZ);  /* set the local ip address in the arp pkt */

    movestr( nulladdr, arppktp->remethad, E_ADD_SIZ); /*set the remote ethernet address in the arp pkt to nulls
                                                                                                    */

    movestr( host1, arppktp->remprotoad, IP_ADD_SIZ);   /* set the remote ip address in the arp pkt to the host's
                                                                                            ip address */

    retrycnt = 0;           /* prepare to send the arp packet five times */

    do
    {       /* send the arp request and wait for an arp table entry
```

```
                to be created by the arp reply from the host */

                status = sendpkt( ( unsigned char *) ethhdrp, E_MN_PSIZ);
                retrycnt++;
                delay( 1000);                   /* wait 1 sec. */
                getpkts();              /* process any packets on the packet queue */

        } while( ( ( arpaddrp = srcharptbl( host1)) == NULL) && ( status >= 0) && ( retrycnt < 5));
                                                /* loop if the reply hasn't arrived, there were no errors on sending
                                                the arp request packet, and we haven't timed out */

        free( ( void *) ethhdrp);               /* free the arp packet's memory */

        if( status < 0)
        {       /* was an arp table entry for the host found? */

                fprintf( stderr, "getethaddr: can't send arp packet, error %d\n", status);

                return( NULL);                          /* no entry, return error */
        }

        if( ( retrycnt >= 5) && ( arpaddrp == NULL))            /* did we time out? */
                return( NULL);                          /* yes, return error */

        if( host != host1)
        {                       /* did we get the host's address? */
                                /* no, we got the gateway's */
                if( ( arpaddrp = getethaddr( host)) == NULL)
                {       /* can we get the host's ethernet address? */

                        fprintf( stderr, "getethaddr: can't get host address after getting the gateway's\n");
                        return( NULL);                  /* no, return error */
                }

                /* yes, return it */
        }
        return( arpaddrp);      /* return a pointer to the host's ethernet address */
}

// END GETRETHAD.C
```

```
/*---- getpkts() ----------------------------------------*/
/* Remove any packets from the packet driver packet queue and dispatch
them to the appropriate processing routines.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>              /* for NULL */
#include   <dos.h>                        /* for FP_OFF(), etc. */

extern struct bufferblk * far bufroot;
extern void far buffmgr( void);

int getpkts( void)
{
        extern int arphandle;
        extern int iphandle;
        extern struct dvrinfo pdvrinfo;
        int arphdlr( struct bufferblk far *),
        iphdlr( struct bufferblk far *),
        relbuffer( struct bufferblk far *);
        int initdvr( struct dvrinfo *, unsigned char *, int, char far *);
        char *prdvrerr( int);
        struct bufferblk far *bufblkp;
        unsigned char far *bp;
        int status;

        if( arphandle < 0)
        {       /* is an arp handle active? */
                int arptype;            /* no, activate one */
                arptype = BYTESWAP( E_TYPE_ARP);
                if( ( arphandle = initdvr( &pdvrinfo, ( unsigned char *) &arptype, sizeof( arptype), ( char far *)buffmgr)) < 0)
                {
                        fprintf( stderr, "getpkts: can't open arp type, error %s\n",  prdvrerr( arphandle));
                        return( ERROR);
                }
        }

        if( bufroot == NULL)  /* is the queue empty? */

        return( 0);                                             /* yes, return 0 */
        bufblkp = ( struct bufferblk far *) &bufroot;           /* set segment of buffer pointer correctly */
        while( bufroot != NULL)
        {       /* loop while packets are on the queue */

                FP_OFF( bufblkp) = ( unsigned int) bufroot;  /* point to first buf */
                bufroot = bufblkp->nxtblockp;                /* remove the first buffer */
                if( bufblkp->handle == arphandle)
                {       /* is it an arp packet? */

                        status = arphdlr( bufblkp);          /* yes, send it to the arp handler */
                        if( status < 0)                      /* any errors on packet processing? */
                                return( status);             /* return the status of the arp hdlr */
                }
                else
                {                                       /* not an arp packet */
                        if( bufblkp->handle == iphandle)
                        {       /* is it an ip packet? */

                                if( iphdlr( bufblkp) < 0)
                                {       /* yes, send it to the ip handler, processed ok? */

                                        fprintf( stderr, "getpkts: error from iphdlr\n");
                                }
                                /* ip packet processed or released as necessary */
                        }

                        else
                        {                               /* not an arp or ip packet, throw it away */

                                fprintf( stderr, "getpkts: got non-arp/non-ip packet\n");
```

```
                                        bufblkp->usedflag = 0;                  /* release the buffer */
                                }
                        }
                }
                return( 1);             /* return "packets successfully processed" status */
}

// END GETPKTS.C

/*---- qmgrglob.c ----------------------------------------*/
/*---- global variables ----------------------------------*/

#include    <stdio.h>               /* for NULL */

int arphandle = ( -1);

struct arpentry *arptblqptr = NULL;


// END QMGRGLOB.C.
```

```
/*---- sendarprep() ----------------------------------------*/
/*  Given a far pointer to an ARP request packet, send an ARP response.

*/

#include     "\gary\snmp\snmp_src\pd\pdnet.h"
#include     <malloc.h>          /* for malloc(), etc. */
#include     <stdio.h>           /* for fprintf(), etc. */

int sendarprep( struct arppkt far *arppktp)
{
          extern struct dvrinfo pdvrinfo;
          void *malloc( size_t);
          void free( void *);
          int sendpkt( unsigned char *, int);
          int movestr( unsigned char *, unsigned char *, int);
          int fmovestr( unsigned char far *, unsigned char far *, int);
          struct ethhdr *ethhdrp;
          struct arppkt *app;
          int status;

          if( ( ethhdrp = ( struct ethhdr *) malloc( E_MN_PSIZ)) == NULL)
          {

                    fprintf( stderr, "sendarprep: canot get a memory buffer\n");
                    return( ERROR);
          }

          app = ( struct arppkt *) ( ethhdrp + 1);

          fmovestr( ( unsigned char far *) arppktp->locethad,  ( unsigned char far *) ethhdrp->destaddr, E_ADD_SIZ);

          movestr( pdvrinfo.locethaddr, ethhdrp->srcaddr, E_ADD_SIZ);

          ethhdrp->type = BYTESWAP( E_TYPE_ARP);

          app->hdwrtype = BYTESWAP( ARP_HDWR_ETH);
          app->prototype = BYTESWAP( E_TYPE_IP);
          app->operation = BYTESWAP( ARP_OP_REP);
          app->hdwradlen = E_ADD_SIZ;
          app->protoadlen = IP_ADD_SIZ;

          movestr( pdvrinfo.locethaddr, app->locethad, E_ADD_SIZ);
          movestr( pdvrinfo.locipaddr, app->locprotoad, IP_ADD_SIZ);
          fmovestr( ( unsigned char far *) arppktp->locethad,   ( unsigned char far *) app->remethad, E_ADD_SIZ);
          fmovestr( ( unsigned char far *) arppktp->locprotoad,  ( unsigned char far *) app->remprotoad, IP_ADD_SIZ);
          if( ( status = sendpkt( ( unsigned char *) ethhdrp, E_MN_PSIZ)) >= 0)
                    status = 0;

          else
                    fprintf( stderr, "sendarprep: canot send arp packet, error %d\n", status);

          free( ( void *) ethhdrp);

          if( status >= 0)
          {
                    return( addarpent( arppktp));
          }
          else
                    return( status);
};

// END ASARPREP.C
```

```
/*---- showarptbl() ----------------------------------------------*/
/*  Show the contents of the ARP table.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                /* for NULL, fprintf(), etc. */

int showarptbl( void)
{
        extern struct arpentry *arptblqptr;
        struct arpentry *aep;
        int i;

        aep = arptblqptr;

        while( aep != NULL)
        {
                fprintf( stderr, "Ethernet address ");

                for( i = 0; i < E_ADD_SIZ; i++)
                        fprintf( stderr, "%02x:", aep->ethaddr[ i]);

                fprintf( stderr, "\n\tInternet address ");

                for( i = 0; i < IP_ADD_SIZ; i++)
                        fprintf( stderr, "%u.", ( unsigned int) aep->ipaddr[ i]);

                putc( '\n', stderr);
                aep = aep->nxtentryp;
        }
        return( 0);
}

// END SHARPTBL.C
```

```
/*--- srcharptbl() ----------------------------------------*/
/*  Given a host's IP address, look for a matching Ethernet address in
the ARP tables.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                /* for fprintf(), etc. */

unsigned char *srcharptbl( unsigned char *host)
{
        extern struct arpentry *arptblqptr;
        int cmpstr( unsigned char *, unsigned char *, int);
        struct arpentry *arpentp;
        int cmpstatus;

        if( arptblqptr == NULL)                 /* are there any arp entries? */
                return( NULL);                  /* no, return NULL */

        arpentp = arptblqptr;                   /* point at the first entry */
                                                /* loop while ip addresses don't match and there are entries */

        while( ( arpentp != NULL) && ( cmpstatus =  cmpstr( arpentp->ipaddr, host, IP_ADD_SIZ) != 0))
        {
                arpentp = arpentp->nxtentryp;
        }

        if( cmpstatus != 0)                     /* found a matching ip addr? */
        return( NULL);                          /* no, return NULL */


        return( arpentp->ethaddr);              /* return a pointer to the eth addr */
}

// END SRCHARPT.C
```

```
/*----- addbuf2q() ----------------------------------------------*/
/* Given a far pointer to a bufferblk and a pointer to a bufferblk
queue, add the bufferblk to the queue.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */
#include   <dos.h>                                   /* for FP_SEG(), etc. */

int addbuf2q( struct bufferblk far *bufferblkp, struct bufferblk far **queue)
{
        struct bufferblk far *bbp;

        bufferblkp->nxtblockp = NULL;
        bbp = *queue;

        if( FP_OFF( bbp) == NULL)
        {
                *queue = bufferblkp;
                return( 0);
        }

        if( FP_SEG( bbp) != FP_SEG( bufferblkp))
        {

                fprintf( stderr, "addbuf2q: buffer and queue have different segments\n");
                return( ERROR);
        }
        while( bbp->nxtblockp != NULL)
                FP_OFF( bbp) = ( unsigned int) bbp->nxtblockp;

        bbp->nxtblockp = ( struct bufferblk *) FP_OFF( bufferblkp);
        return( 0);
}

// END ADDBUF2Q.C
```

```
/*----- atonetaddr() -----------------------------------------------*/
/*  A function to convert a string of characters to a 32 bit long
binary number in network order containing the Internet address of
the characters passed.

*/

#include   "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"
#include   <stdio.h>              /* for NULL, etc. */

#define    ERROR                  ( -1)

long atonetaddr( addr)

char *addr;
{
        int atoi( const char *);
        union {
        long netaddr;
        unsigned char bytes[4];
        } u;
        char buffer[BUFLEN],
        *secnum,
        *thirdnum,
        *fourthnum;
        int i;
        secnum = thirdnum = fourthnum = NULL;
        for( i = 0 ; addr[i] != '\0' ; i++)
        {       /* find the length of the string and move it to the buffer */
                buffer[i] = addr[i];    /* move the character to the buffer */
                if( buffer[i] == '\.')
                {
                        buffer[i] = '\0';

                        if( secnum == NULL)
                                secnum = &( buffer[i+1]);

                        else if( thirdnum == NULL)
                                thirdnum = &( buffer[i+1]);

                                else if( fourthnum == NULL)
                                        fourthnum = &( buffer[i+1]);
                }
                if( i == ( BUFLEN - 1))
                        return( ( long)( ERROR));
        }

        if( addr[i] != '\0')                    /* did the loop end correctly? */
                return( ( long)( ERROR - 1));   /* no, error */

        buffer[i] = '\0';                       /* terminate the last string */
        u.bytes[0] = ( unsigned char)atoi( buffer);

        u.bytes[1] = ( unsigned char)atoi( secnum);

        u.bytes[2] = ( unsigned char)atoi( thirdnum);

        u.bytes[3] = ( unsigned char)atoi( fourthnum);
        return( u.netaddr);
}

// END ATONETAD.C
```

```
/*---- calicmpcksm() ----------------------------------------*/
/* Given a far pointer to an IP packet containing an ICMP packet,
calculate and return its checksum.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"          /* get common structures */
#include   <stdio.h>          /* for fprintf(), etc. */

unsigned int calicmpcksm( struct iphdr far *iphdrp)
{
        struct icmphdr far *icmphdrp;
        unsigned int far *ip;
        unsigned long cksum, li;
        int intcount, flag, i;

        intcount = ( int) ( F_BYTESWAP( iphdrp->ippktlen) - F_IP_IHL( iphdrp->ipver_hl));

        if( intcount == (( intcount >> 1) << 1))        /* is intcount even? */
                flag = 0;                               /* yes, clear the flag */
        else                                            /* intcount is odd */
                flag = 1;                               /* set the flag */

        intcount = intcount >> 1;          /* get the number of integers to checksum */
                                           /* find the start of the icmp header and start calculating */

        icmphdrp = ( struct icmphdr far *) ( ( unsigned char far *) iphdrp +  F_IP_IHL( iphdrp->ipver_hl));

                /* calculate the checksum of the icmp packet */
        for(i = 0, ip = ( unsigned int far *) icmphdrp,cksum = 0L;i < intcount;i++)

                if( &( ip[ i]) != ( unsigned int far *) &( icmphdrp->icmpxsum))

                        /* is the field to be added not the icmp checksum field? */

                        cksum += ( unsigned long) ip[ i];               /* right, add it */

        /* otherwise, don't add it */

                if( flag != 0)          /* is the a last byte to add to the checksum? */

                        cksum += ( unsigned long) ( *( ( unsigned char far *) &( ip[ i])));

        /* yes, add it properly */

                li = cksum >> 16;                   /* get the bits above the lowest 16 */
                cksum += li;                                /* add them to the lowest of the low 16 */
                cksum -= li << 16;                  /* subtract them from the checksum */

                return( ~( ( unsigned int) cksum));/* return the one's-complement of the calculated value */
}

// END CALICMPC.C
```

```
/*---- calipcksm() ----------------------------------------------*/
/*  Given a far pointer to an IP packet header, calculate the checksum
for the IP header.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */
/*#include <dos.h>                                  /* for FP_SEG(), etc. */

unsigned int calipcksm( struct iphdr far *iphdrp)
{
        unsigned int far *ip;
        unsigned long cksum,  li;
        int intcount,i;
        unsigned char uc;

        intcount = ( int) ( F_IP_IHL( iphdrp->ipver_hl) >> 1);
                                                  /* get header length in 16 bit words */
        /* start checksumming the ip header */

        for( i = 0, ip = ( unsigned int far *) iphdrp,cksum = 0L;i < intcount;i++)
                if( &( ip[ i]) != ( unsigned int far *) &( iphdrp->hdrxsum))

                        /* not pointing at the ip checksum field? */
                        cksum += ( unsigned long) ip[ i];  /* right, checksum it */

        /* otherwise, don't checksum it */

        li = cksum >> 16;               /* get any bits above the lowest 16 */

        cksum += li;                    /* add them to the lowest of the low 16 */
        cksum -= li << 16;              /* subtract them from the checksum */

        return( ~( ( unsigned int) cksum));/* return the one's-complement
                                              of the calculated value */
}

// END CALIPCK.C
```

```
/*---- caltcpcksm() ----------------------------------------*/
/* Given a far pointer to an IP packet header, calculate the checksum of
the TCP packet in the IP packet.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>          /* for fprintf() */
/*#include <dos.h>                    /* for FP_SEG(), etc. */

unsigned int caltcpcksm( struct iphdr far *iphdrp)
{
        struct tcphdr far *tcphdrp;
        unsigned int far *ip;
        unsigned long cksum, li;
        int intcount, flag, i;

        ip = ( unsigned int far *) iphdrp->srcaddr;          /* point at the ip source address */

        cksum = ( unsigned long) ip[ 0] + ( unsigned long) ip[ 1] + ( unsigned long) ip[ 2] + ( unsigned long) ip[ 3];

                /* add the src and dest addresses to the checksum */

        i = ( int) iphdrp->proto;
        cksum += ( unsigned long) ( BYTESWAP( i));          /* add the protocol number */

                /* get the length in bytes of the tcp packet */
        intcount = ( int) ( F_BYTESWAP( iphdrp->ippktlen) - F_IP_IHL( iphdrp->ipver_hl));

        cksum += ( unsigned long) ( BYTESWAP( intcount));          /* add it to the checksum */
        if( intcount == (( intcount >> 1) << 1))        /* is intcount even? */
                flag = 0;                               /* yes, clear the flag */
        else                                            /* intcount is odd */
                flag = 1;                               /* set the flag */

        intcount = intcount >> 1;          /* get the number of integers to checksum */
                                           /* find the start of the tcp header */

        tcphdrp = ( struct tcphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

                /* calculate the checksum for the tcp packet */
        for( i = 0, ip = ( unsigned int far *) tcphdrp; i < intcount; i++)

                if( &( ip[ i]) != ( unsigned int far *) &( tcphdrp->tcpxsum))

                                /* is the field to be added not the tcp checksum field? */
                        cksum += ( unsigned long) ip[ i];  /* right, add it */

        /* otherwise, ignore it */
                if( flag > 0)           /* one last byte to add to the checksum? */
                        cksum += ( unsigned long) ( *( ( unsigned char far *) &( ip[ i]))); /* add the last byte */
                li = cksum >> 16;                   /* get any bits above the lowest 16 */
        cksum += li;                    /* add them to the lowest of the low 16 */
        cksum -= li << 16;              /* subtract them from the checksum */

        return( ~( ( unsigned int) cksum));          /* return the one's-
                                        complement of the calculated value */
}

// END CALTCPCK.C
```

```
/*---- caludpcksm() ----------------------------------------------*/
/* Given a far pointer to an IP packet header, calculate the checksum of
the UDP packet in the IP packet. Revised 5-31-90.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */

unsigned int caludpcksm( struct iphdr far *iphdrp)
{
        struct udphdr far *udphdrp;
        unsigned int far *ip;
        unsigned long cksum, li;
        int intcount, flag, i;

        ip = ( unsigned int far *) iphdrp->srcaddr;          /* point at the ip source address */
        cksum = ( unsigned long) ip[ 0] + ( unsigned long) ip[ 1] + ( unsigned long) ip[ 2] + ( unsigned long) ip[ 3];

                        /* add the src and dest addresses to the checksum */

        i = ( int) iphdrp->proto;

        cksum += ( unsigned long) ( BYTESWAP( i)); /* add the protocol number */

                        /* get the length in bytes of the udp packet */

        intcount = ( int) ( F_BYTESWAP( iphdrp->ippktlen) - F_IP_IHL( iphdrp->ipver_hl));

        cksum += ( unsigned long) ( BYTESWAP( intcount));    /* add it to  the checksum */

        if( intcount == (( intcount >> 1) << 1))        /* is intcount even? */

                flag = 0;                               /* yes, clear the flag */
        else                                            /* intcount is odd */
                flag = 1;                               /* set the flag */

        intcount = intcount >> 1;           /* get the number of integers to checksum */
                                            /* find the start of the udp header and start calculating */

        udphdrp = ( struct udphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

                /* calculate the checksum of the udp packet */

        for( i = 0, ip = ( unsigned int far *) udphdrp; i < intcount; i++)
                if( &( ip[ i]) != ( unsigned int far *) &( udphdrp->udpxsum))

                /* is the field to be added not the udp checksum field? */
                        cksum += ( unsigned long) ip[ i];  /* right, add it */
                                /* otherwise, don't add it */
                if( flag != 0)          /* is the a last byte to add to the checksum? */
                cksum += ( unsigned long) ( *( ( unsigned char far *) &( ip[ i])));
                                /* yes, add it properly */
                li = cksum >> 16;               /* get the bits above the lowest 16 */

        cksum += li;                    /* add them to the lowest of the low 16 */
        cksum -= li << 16;              /* subtract them from the checksum */

        return( ~( ( unsigned int) cksum));  /* return the one's-complement of the calculated value */
}

// END CALUDPCK.C
```

```
/*---- cktcpskt() ----------------------------------------*/
/* Look at the given packet and check to see if it's fhost,
fsocket, and lsocket are on the open socket list. For those that
are kept, set 'handle' in the buffer block to be the handle in the
socket block and return zero. Otherwise return nonzero.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>          /* for fprintf() */
#include   <dos.h>                    /* for FP_SEG(), etc. */

int cktcpskt( struct bufferblk far *bufferblkp)
{
        extern struct sockdata *socketroot;
        extern struct bufferblk far *tcpqptr;
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        struct sockdata *sdp;
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct tcphdr far *tcphdrp;
        int status;

        if( socketroot == NULL)
        {               /* any open sockets? */
                struct bufferblk far *bbp;
                bbp = tcpqptr;                          /* no, release all udp queue buffers */
                while( FP_OFF( tcpqptr) != NULL)
                {
                        FP_OFF( tcpqptr) = ( unsigned int) bbp->nxtblockp;
                        bbp->usedflag = 0;
                        bbp = tcpqptr;
                }
                return( ERROR);
        }

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;
        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);
        tcphdrp = ( struct tcphdr far *) ( ( unsigned char far *)iphdrp + F_IP_IHL( iphdrp->ipver_hl));
        sdp = socketroot;

        do
        {               /* loop while remote host, remote socket, and local socket don't match */
                if( sdp->protocol == IP_PROTO_TCP)
                {
                        if( ( tcphdrp->srcport == BYTESWAP( sdp->fsocket)) &&
                                ( tcphdrp->destport == BYTESWAP( sdp->lsocket)))
                        {
                                                /* sockets matched? */
                                if( sdp->fhost == 0L)
                                {               /* yes, foreign host is 0? */
                                        status = 0;                     /* yes, keep this packet */
                                }
                                else
                                {                               /* foreign host is not 0 */
                                        if( ( status = fcmpstr( ( unsigned char far *) &( sdp->fhost),
                                                ( unsigned char far *) iphdrp->srcaddr, IP_ADD_SIZ)) != 0)
                                        {
                                                /* foreign hosts match? */
                                                sdp = sdp->nxtstrcp;    /* no, go to next socket */
                                        }
                                }
                        }
                        else
                        {       /* socket numbers didn't match, go to next socket */
                                status = ( -1);                 /* set loop variable */
                                sdp = sdp->nxtstrcp;
                        }
                }
                else
                {               /* socket is not a tcp socket, go to next */
```

```
                        sdp = sdp->nxtstrcp;
                        status = (-1);
                }
        } while( ( sdp != NULL) && ( status != 0));

        if( status != 0)
        {       /* any matching socket for that packet? */

                fprintf( stderr, "cktcpskt: couldn't find matching tcp socket\n");

                return( ERROR);
        }

        bufferblkp->handle = sdp->nd;     /* set buffer handle = socket nd */

        return( 0);             /* found matching tcp socket */
}

// END CKTCPSKT.C
```

```
/*----- ckudpskt() ----------------------------------------*/
/* Look at the given packet and check to see if it's fhost,
fsocket, and lsocket are on the open socket list. For those that
are kept, set 'handle' in the buffer block to be the handle in the
socket block and return zero. Otherwise return nonzero.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */
#include   <dos.h>                                   /* for FP_SEG(), etc. */

int ckudpskt( struct bufferblk far *bufferblkp)
{
        extern struct sockdata *socketroot;
        extern struct bufferblk far *udpqptr;
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        struct sockdata *sdp;
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct udphdr far *udphdrp;
        int status;

        if( socketroot == NULL)
        {               /* any open sockets? */
                struct bufferblk far *bbp;
                bbp = udpqptr;                          /* no, release all udp queue buffers */
                while( FP_OFF( udpqptr) != NULL)
                {
                        FP_OFF( udpqptr) = ( unsigned int) bbp->nxtblockp;
                        bbp->usedflag = 0;
                        bbp = udpqptr;
                }
                return( ERROR);
        }

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;
        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);
        udphdrp = ( struct udphdr far *) ( ( unsigned char far *)iphdrp + F_IP_IHL( iphdrp->ipver_hl));
        sdp = socketroot;

        do
        {               /* loop while remote host, remote socket, and local socket don't match */

                if( sdp->protocol == IP_PROTO_UDP)
                {       /* is this a UDP socket? */

                        if( sdp->fhost == 0L)
                        {       /* yes, accept pkts from any host? */

                                if( sdp->fsocket == 0)
                                {       /* yes, accept pkts from any port? */

                                        if( udphdrp->destport == BYTESWAP( sdp->lsocket))
                                        {
                                                /* yes, is the dest port open? */
                                                status = 0; /* yes, accept the packet */
                                        }
                                        else
                                        {               /* the dest port is not open */
                                                status = ( -1);                 /* reject this socket */
                                                sdp = sdp->nxtstrcp;    /* go to next socket */
                                        }
                                }
                                else
                                {               /* accept packets from a specific port */
                                if( ( udphdrp->srcport == BYTESWAP( sdp->fsocket)) &&
                                        ( udphdrp->destport == BYTESWAP( sdp->lsocket)))
                                {
                                                /* do the remote and local ports match? */
                                        status = 0; /* yes, accept this packet */
```

```
                                        }
                                        else
                                        {               /* the ports don't match */
                                                status = ( -1);                 /* reject this socket */
                                                sdp = sdp->nxtstrcp;            /* go to next socket */
                                        }
                                }
                        }
                        else
                        {       /* accept packets from a specific host only */
                                if( ( status = fcmpstr( ( unsigned char far *) &( sdp->fhost), ( unsigned char far *) iphdrp->srcaddr,
                                                        IP_ADD_SIZ)) != 0)
                                {
                                                        /* remote host matches? */
                                        unsigned char *bytep;
                                        status = ( -1);                 /* no, reject this packet */
                                        sdp = sdp->nxtstrcp;    /* go to next socket */
                                }
                                else
                                {                       /* remote host matches */
                                        if( sdp->fsocket == 0)
                                        {       /* accept any remote port? */
                                                if( udphdrp->destport == BYTESWAP( sdp->lsocket))
                                                {
                                                /* yes, destination port open? */
                                                        status = 0; /* yes, accept this packet */
                                                }
                                                else
                                                {               /* destination port not open */
                                                        status = ( -1);         /* reject this packet */
                                                        sdp = sdp->nxtstrcp;    /* go to next skt */
                                                }
                                        }
                                        else
                                        {       /* remote and local ports must match */
                                                if( ( udphdrp->srcport == BYTESWAP( sdp->fsocket)) &&
                                                                        ( udphdrp->destport ==
                                                                                BYTESWAP( sdp->lsocket)))
                                                {
                                                /* local and remote ports match? */
                                                        status = 0;             /* yes, accept this packet */
                                                }
                                                else
                                                {       /* remote and local ports don't match */
                                                        status = ( -1);         /* reject this packet */
                                                        sdp = sdp->nxtstrcp;    /* go to next skt */
                                                }
                                        }
                                }
                        }
                        else
                        {                               /* not a udp socket, go to the next one */
                                sdp = sdp->nxtstrcp;
                                status = ( -1);
                        }
        } while( ( sdp != NULL) && ( status != 0));
        if( status != 0)
        {       /* any matching socket for that packet? */
                return( ERROR);
        }
                                /* found a matching socket */
        bufferblkp->handle = sdp->nd;   /* set buffer handle = socket nd */
        return( 0);                     /* found matching udp socket */
}
// END CKUDPSKT.C

/*---- culltcpq() ----------------------------------------*/
/* Remove any packets from the TCP queue that don't have active
socket handles associated with them.

*/
```

```
#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */
#include   <dos.h>                                   /* for FP_SEG(), etc. */

int culltcpq( void)
{
        extern struct bufferblk far *tcpqptr;
        unsigned int calipcksm( struct iphdr far *);
        unsigned int caltcpcksm( struct iphdr far *);
        struct bufferblk far *bbp;
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct tcphdr far *tcphdrp;
        unsigned int pktcksum;


        if( FP_OFF( tcpqptr) == NULL)
        {                       /* are there packets queued? */
                return( 0);                     /* no, return */
        }

        while( FP_OFF( tcpqptr) != NULL)
        {       /* while there are packets queued, loop */
                bbp = tcpqptr;                                  /* point at a packet buffer */
                FP_OFF( tcpqptr) = ( unsigned int) bbp->nxtblockp;      /* remove that buffer from the queue */
                bbp->usedflag = 0;                              /* free the packet buffer */
        }

        return( 0);
}

// END CULLTCPQ.C
```

```
/*---- culludpq() ----------------------------------------------*/
/*  Look at the packets on the UDP queue and throw those away that
are not being listened for.  For those that are kept, set 'handle'
in the buffer block to be the handle in the socket block.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */
#include   <dos.h>                                   /* for FP_SEG(), etc. */

int culludpq( void)
{
        extern struct sockdata *socketroot;
        extern struct bufferblk far *udpqptr;
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        struct bufferblk far *bbp, far *prevbbp;
        struct sockdata *sdp;
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct udphdr far *udphdrp;
        unsigned char far *fbytep;

        if( socketroot == NULL)
        {               /* any open sockets? */
                bbp = udpqptr;                      /* no, release all buffers */
                while( FP_OFF( udpqptr) != NULL)
                {
                        FP_OFF( udpqptr) = ( unsigned int) bbp->nxtblockp;
                        bbp->usedflag = 0;
                        bbp = udpqptr;
                }

                return( 0);
        }

        bbp = udpqptr;
        while( FP_OFF( bbp) != NULL)
        {
                int status;
                ethhdrp = ( struct ethhdr far *) bbp->data;
                iphdrp = ( struct iphdr far *) ( ethhdrp + 1);
                udphdrp = ( struct udphdr far *) ( ( unsigned char far *)iphdrp +
                                                        F_IP_IHL( iphdrp->ipver_hl));
                sdp = socketroot;
        do
        {               /* loop while remote host, remote socket, and
                                local socket don't match */
                if( ( udphdrp->srcport == BYTESWAP( sdp->fsocket)) &&
                                                ( udphdrp->destport == BYTESWAP( sdp->lsocket)))
                {
                                                /* sockets matched? */
                        if( sdp->fhost == 0L)
                        {                       /* yes, foreign host is 0? */
                                status = 0;     /* yes, keep this packet */
                        }
                        else
                        {                               /* foreign host is not 0 */
                                if( ( status = fcmpstr( ( unsigned char far *) &( sdp->fhost),
                                                        ( unsigned char far *) iphdrp->srcaddr,
                                                                        IP_ADD_SIZ)) != 0)
                                {
                                        /* foreign hosts match? */
                                        sdp = sdp->nxtstrcp;    /* no, go to next socket */
                                }
                        }
                }
                else
                {       /* socket numbers didn't match, go to next socket */
                        status = ( -1);         /* set loop variable */
                        sdp = sdp->nxtstrcp;
```

```
                }

        } while( ( sdp != NULL) && ( status != 0));

        if( status != 0)
        {       /* any matching socket for that packet? */

                fprintf( stderr, "culludpq: couldn't find matching socket\n");
                if( bbp == udpqptr)
                {                       /* no, is this the first buffer blk? */
                                        /* yes, change udpqptr */
                        FP_OFF( udpqptr) = ( unsigned int) bbp->nxtblockp;
                        bbp->usedflag = 0;
                        prevbbp = bbp = udpqptr;
                }
                else
                {                                       /* not first buffer blk */
                        prevbbp->nxtblockp = bbp->nxtblockp;        /* change prev blk ptr */
                        bbp->usedflag = 0;                          /* release the buffer */
                        FP_OFF( bbp) = ( unsigned int) prevbbp->nxtblockp;
                                                                    /* point to next buffer blk */
                }
        }
        else
        {                               /* yes, socket info matched completely */

                bbp->handle = sdp->nd;                      /* set buffer handle */
                prevbbp = bbp;                              /* move pointers to next buffer blks */
                FP_OFF( bbp) = ( unsigned int) bbp->nxtblockp;
        }
        }
        return( 0);
}

// END CULLUDPQ.C
```

```
/*----- endudp() ----------------------------------------------*/
/*  Turn off all UDP functions by releasing all network descriptors,
and disabling packet driver acceptance of IP and ARP packets.

*/

int endudp( void)
{
        extern int arphandle, iphandle;

        int udpreleaseall( void);
        int relhandle( int);
        udpreleaseall();

        if( arphandle >= 0)
                relhandle( arphandle);

        arphandle = ( -1);

        if( iphandle >= 0)
                relhandle( iphandle);

        iphandle = ( -1);

        return( 0);
}

// END ENDUDP.C
```

```
/*----- getudpnd() ----------------------------------------*/
/*  Given a foreign host, foreign socket, and local socket, get a
network descriptor to send UDP packets on.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"
#include   <stdio.h>                /* for NULL, fprintf(), etc. */

int getudpnd( long fhost, unsigned int fsocket, unsigned int lsocket, unsigned long option)
{
        extern int verbose;

        int net_getdesc( void);
        int net_connect( int, int, struct addr *);
        void *malloc( unsigned int);
        void free( void *);
        void pneterror( char *);

        extern struct sockdata *socketroot;
        extern int neterrno;
        extern int netsuberrno;

        struct sockdata *sockdatap;
        struct addr *sockinfop;

        int status;

        sockdatap = socketroot;          /* find the beginning of the chain */

        if( sockdatap == NULL)
        {                        /* is there no chain? */
                                 /* yes, can we get memory for a structure? */
                if( ( sockdatap = socketroot = ( struct sockdata *)malloc( sizeof( struct sockdata))) == NULL)
                {
                        fprintf( stderr, "getudpnd: malloc error on first structure\n");
                        return( ERROR);          /* got memory for next structure? no, error */
                }
        }
        else
        {                        /* there is a chain */
                while( sockdatap->nxtstrcp != NULL)
                {        /* find the end of the chain */
                         /* is a socket already open? */

                        if( ( sockdatap->fhost == fhost) && ( sockdatap->fsocket == fsocket) &&
                                                                        ( sockdatap->lsocket == lsocket))
                        {
                                return( sockdatap->nd);          /* yes, return its net desc */
                        }

                        sockdatap = sockdatap->nxtstrcp;  /* go to next socket */
                }

                if( ( sockdatap->nxtstrcp = ( struct sockdata *)malloc( sizeof( struct sockdata)))== NULL)
                {
                        fprintf( stderr, "getudpnd: malloc error in middle of chain\n");
                        return( ERROR - 1);          /* got memory for next structure?  no, error */
                }
                sockdatap = sockdatap->nxtstrcp;  /* point to the new structure */
        }                                         /* initialize the structure */

        if( verbose == 257)
                fprintf( stderr, "getudpnd: added a sockdata struct at %04x\n",

        sockdatap);
        sockdatap->nxtstrcp = NULL;
        sockdatap->reqid = ( long)( -1);
        sockdatap->netoption = option;
        sockinfop = ( struct addr *) &( sockdatap->fhost);
```

```
sockinfop->fhost = fhost;
sockinfop->fsocket = fsocket;
sockinfop->lsocket = lsocket;

if( ( status = net_connect( (-1), ( int)DGRAM, sockinfop)) >= 0)
{
                                                    /* got a valid net descriptor? */
        sockdatap->nd = status;                     /* yes, save it */
                                                    /* set timeout value in millisecs */
        if( option == TIMEOUTFL)
        {
                if( verbose == 257)
                        fprintf( stderr, "getudpnd: option is timeout\n");

                if( ( status = set_option( sockdatap->nd, ( int)DGRAM,
                                                    ( int)NET_OPT_TIMEOUT, ( char far *)option, ( int) 0)) < 0)
                {
                        fprintf( stderr, "getudpnd: timeout set_option error %d\n", status);
                        return( ERROR - 2);
                }
        }
                /* set non-blocking option */
        else if( option == NOWAITINGFL)
        {
                if( verbose == 257)
                        fprintf( stderr, "getudpnd: option is non-blocking\n");

                if( ( status = set_option( sockdatap->nd, ( int)DGRAM,   ( int)NET_OPT_NONBLOCKING,
                                                                        ( char far *)option, ( int)0)) < 0)
                {
                        fprintf( stderr, "getudpnd: nowait set_option error %d\n", status);
                        pneterror( "getudpnd: ");
                        return( ERROR - 2);
                }
        }

        return( sockdatap->nd);                     /* return the net desc. */

}

else
{       /* didn't get a valid net descriptor, clean up */
        fprintf( stderr, "getudpnd: can't get an nd, error %d\n", neterrno);

        if( ( neterrno == NET_ERR_HOST_UNREACHABLE) || ( neterrno == NET_ERR_ICMPMSG))
                fprintf( stderr, "\tnetsuberrno = %d\n", netsuberrno);

        pneterror( "getudpnd: ");

        if( sockdatap == socketroot)
        {                       /* release the sockdata structure associated with 'nd' */
                if( sockdatap->nxtstrcp == NULL)
                        socketroot = NULL;
                else
                        socketroot = sockdatap->nxtstrcp;
        }
        else
        {       /* not the first in the sockdata chain */
                struct sockdata *sdp;
                sdp = socketroot;

                while( ( sdp != NULL) && ( sdp->nxtstrcp != sockdatap))
                        sdp = sdp->nxtstrcp;


                if( sdp == NULL)
                {
                        fprintf( stderr, "getudpnd: sockdata chain corrupted\n");
                        return( ERROR - 3);
                }
                if( sockdatap->nxtstrcp == NULL)
```

```
                                        sdp->nxtstrcp = NULL;
                                else
                                        sdp->nxtstrcp = sockdatap->nxtstrcp;
                        }
                        free( ( void *) sockdatap);
                        if( verbose == 257)
                                fprintf( stderr, "getudpnd: freed a sockdata blk at %04x\n", sockdatap);
                        return( ERROR - 4);
                }
}

// END GETUDPND.C
```

```
/*---- getudppkt() ----------------------------------------*/
/* Get a UDP packet given the network descriptor of the associated
connection, the buffer in which to put the data, the length of the
buffer, and any flags.  Return the status of the read from the net
on success or failure, or ERROR on failure.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                              /* for NULL, fprintf(), etc. */
#include   "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"

int getudppkt( int nd, unsigned char *buffer, unsigned int buflen, unsigned int flags)
{
        extern int neterrno;
        extern int verbose;
        int net_readfrom( int, char *, unsigned, struct addr *, unsigned);
        void pneterror( char *);
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap;
        struct addr *sockinfop;
        int status;

        if( socketroot == NULL)
        {       /* any net descriptors allocated? */
                fprintf( stderr, "getudppkt: no descriptors allocated\n");
                return( ERROR);                 /* no, error */
        }
        sockdatap = socketroot;                 /* yes, find the right one */
        status = 0;

        while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
        {
                sockdatap = sockdatap->nxtstrcp;    /* walk the chain while the right structure hasn't been found and
                                                       there are more struct */
                status++;
        }
        if( sockdatap->nd != nd)
        {       /* found the right structure? */
                fprintf( stderr, "getudppkt: net descriptor mismatch\n");
                fprintf( stderr, "\t%d sockets open\n", status);
                fprintf( stderr, "\tsocketroot = %04x\n", socketroot);
                return( ERROR - 1);             /* no, error */
        }
        sockinfop = ( struct addr *) &( sockdatap->fhost);      /* yes, point to it */
                                                                /* read from the UDP packet queue */
        if( ( status = net_readfrom( nd, buffer, buflen, sockinfop, flags)) < 0)
        {
                if( ( sockdatap->netoption == NOWAITINGFL) && ( neterrno == NET_ERR_WOULD_BLOCK))
                        status = 0;
                else
                {
                        if( ( sockdatap->netoption == TIMEOUTFL) && ( neterrno == NET_ERR_TIMEOUT))
                                status = 0;
                        else
                        {
                                fprintf( stderr, "getudppkt: error %d on \'net_readfrom()\'\n", status);
                                pneterror( "getudppkt: ");
                                status = ERROR - 2;
                        }
                }
        }
        else
                if( verbose == 258)
                        fprintf( stderr, "getudppkt: net_readfrom status was %d\n", status);
        return( status);
}
// END GETUDPPK.C

/*---- icmphdlr() -----------------------------------------*/
/* Given a pointer to a packet buffer containing an ICMP packet,
```

```
respond appropriately to the ICMP message.  If the packet is an ICMP
destination unreachable, time-to-live exceeded, redirect, or echo
message, respond appropriately.  Ignore all other ICMP packets.

*/

#include    "\gary\snmp\snmp_src\pd\pdnet.h"            /* get packet header */
#include    "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"
#include    <stdio.h>                                   /* for fprintf() */
/* #include <dos.h>                                    /* for FP_SEG(), etc. */

int icmphdlr( struct bufferblk far *bufferblkp)
{
        unsigned int calicmpcksm( struct iphdr far *);
        int dstunrch( struct bufferblk far *);
        int redirect( struct bufferblk far *);
        int gotechomsg( struct bufferblk far *);
        int ttlexcd( struct bufferblk far *);
        struct icmphdr far *icmphdrp;
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        int pktlen, i;

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;
        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);    /* point to the ip hdr */

        icmphdrp = ( struct icmphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

        if( icmphdrp->icmpxsum != calicmpcksm( iphdrp))
        {
                bufferblkp->usedflag = 0;
                return( 0);
        }
        switch ( icmphdrp->type)
        {
                case( 3):
                        return( dstunrch( bufferblkp));
                case( 5):
                        return( redirect( bufferblkp));
                case( 8):
                        return( gotechomsg( bufferblkp));
                case( 11):
                        return( ttlexcd( bufferblkp));
                default:
                        bufferblkp->usedflag = 0;                   /* release the buffer */
                return( 0);
        }
}
```

```
/*----- dstunrch() ----------------------------------------------*/
/* Process an ICMP destination unreachable message. Print a message
on the console for the appropriate option.

*/

int dstunrch( struct bufferblk far *bufferblkp)
{
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct icmphdr far *icmphdrp;
        int i;

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;

        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);

        icmphdrp = ( struct icmphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

        switch ( icmphdrp->code)
        {
                case( 0):
                        fprintf( stderr, "icmphdlr: net unreachable for host ");
                        break;
                case( 1):
                        fprintf( stderr, "icmphdlr: host unreachable for host ");
                        break;
                case( 2):
                        fprintf( stderr, "icmphdlr: protocol unreachable for host ");
                        break;
                case( 3):
                        fprintf( stderr, "icmphdlr: port unreachable for host ");
                        break;
                case( 4):
                        fprintf( stderr, "icmphdlr: fragmentation needed for host ");
                        break;
                default:
                        bufferblkp->usedflag = 0;

                return( 0);
        }
        iphdrp = ( struct iphdr far *) ( icmphdrp + 1);

        for( i = 0; i < IP_ADD_SIZ; i++)
                fprintf( stderr, "%d.", ( unsigned int) ( iphdrp->destaddr[ i]));

        putc( '\n', stderr);

        bufferblkp->usedflag = 0;

        return( 0);
}
```

```
/*----- redirect() ----------------------------------------------*/
/* Process an ICMP redirect message. Change the Ethernet address
in the ARP table for the host which is mentioned in the redirect
packet.

*/

int redirect( struct bufferblk far *bufferblkp)
{
        extern struct arpentry *arptblqptr;
        void *malloc( size_t);
        unsigned char *getethaddr( unsigned char *);
        struct arpentry *arpentryp;
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp, far *iphdrp1;
        struct icmphdr far *icmphdrp;
        unsigned char newipaddr[ IP_ADD_SIZ], *newethaddr;
        int status;

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;

        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);

        icmphdrp = ( struct icmphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

        fmovestr( ( unsigned char far *) ( &( icmphdrp->ident)),
         ( unsigned char far *) newipaddr, IP_ADD_SIZ);
        iphdrp1 = ( struct iphdr far *) ( icmphdrp + 1);

        if( ( newethaddr = getethaddr( newipaddr)) == NULL)
        {

                fprintf( stderr, "redirect: can't get new gateway ethernet address\n");
                bufferblkp->usedflag = 0;
                return( ERROR);
        }
        if( ( arpentryp = arptblqptr) == NULL)
        {
                fprintf( stderr, "redirect: no arp table entries, error\n");
                bufferblkp->usedflag = 0;
                return( ERROR);
        }

        status = fcmpstr( ( unsigned char far *) arpentryp->ipaddr, iphdrp1->destaddr, IP_ADD_SIZ);

        while( ( status != 0) && ( arpentryp->nxtentryp != NULL))
        {
                arpentryp = arpentryp->nxtentryp;
                status = fcmpstr( ( unsigned char far *) arpentryp->ipaddr, iphdrp1->destaddr, IP_ADD_SIZ);
        }

        if( status == 0)
        {

                movestr( newethaddr, arpentryp->ethaddr, E_ADD_SIZ);
        }
        else
        {
                struct arpentry *arpp;

                fprintf( stderr, "redirect: couldn't find redirected host address\n");

                if( ( arpp = ( struct arpentry *) malloc( sizeof( struct arpentry)))== NULL)
                {
                        fprintf( stderr, "redirect: couldn't get arp entry memory\n");
                        bufferblkp->usedflag = 0;
                        return( 0);
                }

                arpentryp->nxtentryp = arpp;
                arpp->nxtentryp = NULL;
```

```
                movestr( newethaddr, arpp->ethaddr, E_ADD_SIZ);
                fmovestr( iphdrp1->destaddr, ( unsigned char far *) arpp->ipaddr, IP_ADD_SIZ);
                arpentryp = arpp;
        }

        bufferblkp->usedflag = 0;
        return( 0);
}
```

```
/*---- gotechomsg() ----------------------------------------*/
/* Process an ICMP echo message. Respond by sending an echo reply
packet.

*/

int gotechomsg( struct bufferblk far *bufferblkp)
{
        extern struct dvrinfo pdvrinfo;
        void *malloc( size_t);
        void free( void *);
        int sendpkt( unsigned char *, int);
        int fmovestr( unsigned char far *, unsigned char far *, int);
        int movestr( unsigned char *, unsigned char *, int);
        unsigned int calipcksm( struct iphdr far *);
        unsigned int calicmpcksm( struct iphdr far *);
        int initiphdr( struct iphdr *);
        struct ethhdr far *ethhdrp, *ethhdrp1;
        struct iphdr far *iphdrp,*iphdrp1;
        struct icmphdr far *icmphdrp, *icmphdrp1;
        unsigned char *buffer;
        int bufferlen;

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;
        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);
        icmphdrp = ( struct icmphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

        icmphdrp->type = 0;
        bufferlen = bufferblkp->bufferlen;
        if( ( buffer = ( unsigned char *) malloc( bufferlen)) == NULL)
        {
                fprintf( stderr, "gotechomsg: couldn't get memory\n");
                bufferblkp->usedflag = 0;
                return( ERROR);
        }

        ethhdrp1 = ( struct ethhdr *) buffer;
        iphdrp1 = ( struct iphdr *) ( ethhdrp1 + 1);
        icmphdrp1 = ( struct icmphdr *) ( ( unsigned char *) iphdrp1 + F_IP_IHL( iphdrp->ipver_hl));
        fmovestr( bufferblkp->data, ( unsigned char far *) buffer, bufferlen);
        fmovestr( ethhdrp->srcaddr, ( unsigned char far *) ethhdrp1->destaddr, E_ADD_SIZ);
        fmovestr( iphdrp->srcaddr, ( unsigned char far *) iphdrp1->destaddr,IP_ADD_SIZ);
        bufferblkp->usedflag = 0;
        movestr( pdvrinfo.locethaddr, ethhdrp1->srcaddr, E_ADD_SIZ);
        movestr( pdvrinfo.locipaddr, iphdrp1->srcaddr, IP_ADD_SIZ);
        icmphdrp1->icmpxsum = calicmpcksm( ( struct iphdr far *) iphdrp1);
        iphdrp1->hdrxsum = calipcksm( ( struct iphdr far *) iphdrp1);
        if( sendpkt( buffer, bufferlen) < 0)
        {
                fprintf( stderr, "gotechomsg: couldn't send echo reply packet\n");
        }
        free( ( void *) buffer);
        return( 0);
}
```

```
/*----- ttlexcd() ----------------------------------------*/
/* Process an ICMP time-to-live exceeded message. Print a message
on the console for the appropriate option.

*/

int ttlexcd( struct bufferblk far *bufferblkp)
{
        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        struct icmphdr far *icmphdrp;
        int i;

        ethhdrp = ( struct ethhdr far *) bufferblkp->data;

        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);

        icmphdrp = ( struct icmphdr far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

        switch ( icmphdrp->code)
        {
                case( 0):
                        fprintf( stderr, "icmphdlr: time-to-live exceeded for host ");
                        break;
                case( 1):
                        fprintf( stderr, "icmphdlr: fragment reassembly time exceeded for host ");
                        break;
                default:
                        bufferblkp->usedflag = 0;

                return( 0);
        }

        iphdrp = ( struct iphdr far *) ( icmphdrp + 1);

        for( i = 0; i < IP_ADD_SIZ; i++)
                fprintf( stderr, "%d.", ( unsigned int) ( iphdrp->destaddr[ i]));

        putc( '\n', stderr);
        bufferblkp->usedflag = 0;
        return( 0);
}

// END ICMPHDLR.C

/*----- initiphdr() --------------------------------------*/
/* Given a pointer to a buffer containing an IP header, initialize
it by setting the IP version and header length, the type of service,
the identification, the fragmentation flags, the time-to-live, and
the IP source address.

*/

#include    "\gary\snmp\snmp_src\pd\pdnet.h"

int initiphdr( struct iphdr *iphdrp)
{
   extern struct dvrinfo pdvrinfo;
   int movestr( unsigned char *, unsigned char *, int);
   iphdrp->ipver_hl = 0x45;
   iphdrp->tos = 0;
   iphdrp->id = 0;
   iphdrp->flagment = iphdrp->fragment = 0;
   iphdrp->ttl = (unsigned char) 255;
   movestr( pdvrinfo.locipaddr, iphdrp->srcaddr, IP_ADD_SIZ);
   return( 0);
}
// END INITIPHD.C

/*----- initudp() ----------------------------------------*/
/* Initialize the UDP subroutines by getting the packet driver software
```

```
interrupt number, the packet driver interface class, type, and number,
and reading the IP configuration data.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                        /* for NULL */

int initudp( char *config)
{
        extern struct sockdata *socketroot;
        extern struct routinfo *firstrinfop;
        extern int pdvector;
        extern struct dvrinfo pdvrinfo;
        int getpdint( void);
        int getdvrinfo( struct dvrinfo *);
        int initinfo( void);
        int pdint;

        socketroot = NULL;
        if( ( pdint = getpdint()) < 0)
        {
                fprintf( stderr,
                "initudp: can't get packet driver interrupt, error %d\n", pdint);
                return( ERROR);
        }

        pdvector = pdint;

        if( ( pdint = getdvrinfo( &pdvrinfo)) < 0)
        {
                fprintf( stderr, "initudp: couldn't get driver info, error %d\n", pdint);
                return( ERROR);
        }
        if( initinfo() < 0)                /* initialized driver info? */
                return( ERROR);                                /* no, error */
        return( 0);
}

// END INITUDP.C
```

```
/*----- initudppkt() ----------------------------------------*/
/* Given a pointer to an IP packet buffer, its length, a pointer to
a UDP packet data field, and the length of the data field,
initialize the UDP header, the IP header, and move the data into the
data field. It is assumed that the IP header is complete except for
the packet length field, the protocol, and the header checksum. It
is also assumed that the source and destination UDP ports are
correct.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"

int initudppkt( struct iphdr *iphdrp, int buflen, unsigned char *databufp, int datalen)
{
        int movestr( unsigned char *, unsigned char *, int);
        unsigned int calipcksm( struct iphdr far *);
        unsigned int caludpcksm( struct iphdr far *);
        struct udphdr *udphdrp;
        int pktlen;

        pktlen = sizeof( struct iphdr) + sizeof( struct udphdr) + datalen;

        if( buflen < pktlen)            /* is the packet buffer too small? */
                return( ERROR);                         /* yes, return error */

        if( pktlen > E_MX_PSIZ)                 /* is the final packet too big? */
                return( ERROR);                         /* yes, return error */

                        /* find the start of the udp packet */

        udphdrp = ( struct udphdr *) ( ( unsigned char *) iphdrp + IP_IHL( iphdrp->ipver_hl));

                /* move the data into the udp packet */

        movestr( databufp, ( unsigned char *) ( udphdrp + 1), datalen);
        pktlen = sizeof( struct udphdr) + datalen;                      /* set the udp len */
        udphdrp->pktlen = BYTESWAP( pktlen);                            /* put it in the udp pkt */
        pktlen += IP_IHL( iphdrp->ipver_hl);                            /* add the ip header length */
        iphdrp->ippktlen = BYTESWAP( pktlen);                           /* put it in the ip pkt */
        iphdrp->proto = ( unsigned char) IP_PROTO_UDP;                  /* set the protocol */
        udphdrp->udpxsum = caludpcksm( ( struct iphdr far *) iphdrp);

                                /* get the udp packet's checksum */
        iphdrp->hdrxsum = calipcksm( ( struct iphdr far *) iphdrp);
                                /* get the ip packet's checksum */
        return( 0);
}

// END INITUDPP.C

/*----- ipglob.c --------------------------------------------*/
/*----- global variables ------------------------------------*/

#include   <stdio.h>            /* for NULL */

struct sockdata *socketroot = NULL;         /* temporary */
int iphandle = ( -1);
struct bufferblk far *udpqptr = NULL,
                     far *tcpqptr = NULL;


// END IPGLOB.C
```

```
/*--- iphdlr() ----------------------------------------------*/
/* Process IP packets taken from the packet queue.  If they are
TCP, UDP, or ICMP packets, send a far pointer to the packet buffer
to the appropriate packet handler.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"          /* get packet header */
#include   <stdio.h>                                 /* for fprintf() */
#include   <dos.h>                                   /* for FP_SEG(), etc. */

int iphdlr( struct bufferblk far *bufferblkp)
{
        extern int verbose;
        extern struct dvrinfo pdvrinfo;
        extern struct bufferblk far *udpqptr, far *tcpqptr;

        int addbuf2q( struct bufferblk far *, struct bufferblk far **);
        int fcmpstr( unsigned char far *, unsigned char far *, int);
        unsigned int calipcksm( struct iphdr far *);
        unsigned int caltcpcksm( struct iphdr far *);
        unsigned int caludpcksm( struct iphdr far *);
        int ckudpskt( struct bufferblk far *);
        int cktcpskt( struct bufferblk far *);
        int icmphdlr( struct bufferblk far *);

        struct ethhdr far *ethhdrp;
        struct iphdr far *iphdrp;
        unsigned char far *ucp;
        int status, i;

        FP_SEG( ethhdrp) = FP_SEG( iphdrp) = FP_SEG( bufferblkp);
        FP_OFF( ethhdrp) = ( unsigned int) bufferblkp->data;
                                        /* point to the packet buffer */
        if( ethhdrp->type != BYTESWAP( E_TYPE_IP))
        {       /* is it an IP packet? */
                if( verbose == 259)
                        fprintf( stderr, "iphdlr: got a non-IP packet\n");
                bufferblkp->usedflag = 0;                       /* no, release the buffer */
                return( ERROR);                                 /* return an error */
        }

        iphdrp = ( struct iphdr far *) ( ethhdrp + 1);  /* point to the ip hdr */
        if( calipcksm( iphdrp) != iphdrp->hdrxsum)
        {       /* does the received ip header checksum agree with the calculated one? */

                if( verbose == 259)
                fprintf( stderr, "iphdlr: bad ip checksum, ip packet discarded\n");
                bufferblkp->usedflag = 0;       /* no, release the buffer */
                return( 0);
        }
        else

        if( verbose == 259)
                fprintf( stderr, "iphdlr: ip header checksum was correct\n");

        if( fcmpstr( ( unsigned char far *) pdvrinfo.locipaddr, ( unsigned char far *) iphdrp->destaddr, IP_ADD_SIZ) < 0)
        {       /* is it for my ip addr? */
                if( verbose == 259)
                {
                        fprintf( stderr, "iphdlr: got unwanted IP packet for ");
                        for( i = 0; i < IP_ADD_SIZ; i++)
                                fprintf( stderr, "%d.", ( int) iphdrp->destaddr[ i]);

                        putc( '\n', stderr);
                }

                bufferblkp->usedflag = 0;       /* no, release the buffer */
                return( 0);                     /* return */
        }
```

```
if( verbose == 259)
{
        fprintf( stderr, "iphdlr: got IP packet for ");
        for( i = 0; i < IP_ADD_SIZ; i++)
        fprintf( stderr, "%d.", ( int) iphdrp->destaddr[ i]);
        putc( '\n', stderr);
}
/* point at the header of the protocol inside the ip packet */
ucp = ( unsigned char far *) ( ( unsigned char far *) iphdrp + F_IP_IHL( iphdrp->ipver_hl));

if( iphdrp->proto == IP_PROTO_UDP)
{       /* is it a UDP packet? */

        if( ( ( struct udphdr far *) ucp)->udpxsum != 0)
        {       /* yes, was  the udp checksum transmitted as a 0? */

                if( caludpcksm( iphdrp) != ( ( struct udphdr far *) ucp)->udpxsum)
                {
                        /* no, does the checksum ompute? */
                        if( verbose == 259)
                        fprintf( stderr, "iphdlr: udp checksum bad\n");

                        bufferblkp->usedflag = 0;                       /* no, release the buffer */

                        return( 0);
                }
                else
                {
                        if( verbose == 259)
                        fprintf( stderr, "iphdlr: udp checksum was correct\n");
                }

        }       /* don't compute the udp checksum if it is transmitted as 0 */

        if( ckudpskt( bufferblkp) != 0)
        {       /* is there a socket? */

                if( verbose == 259)
                        fprintf( stderr, "iphdlr: received unwanted udp packet\n");

                bufferblkp->usedflag = 0;       /* no, release the buffer */
        }
        else
        {
                if( ( status = addbuf2q( bufferblkp, &udpqptr)) < 0)
                {       /* can we add it to the udp queue? */

                        if( verbose == 259)
                                fprintf( stderr, "iphdlr: couldn't queue UDP packet, error %d\n", status);

                        bufferblkp->usedflag = 0;       /* no, release the buffer */

                        return( ERROR);                 /* not added to queue, return error */
                }
                if( verbose == 259)
                        fprintf( stderr, "iphdlr: queued UDP packet\n");
        }
}
else
{
        if( iphdrp->proto == IP_PROTO_TCP)
        {       /* is it a TCP packet? */

                if( caltcpcksm( iphdrp) != ( ( struct tcphdr far *)ucp)->tcpxsum)
                {

                        /* yes, tcp checksum is correct? */
                        bufferblkp->usedflag = 0;       /* no, release the buffer */
                        return( 0);
                }
                else
```

```
                {
                        if( verbose == 259)
                        fprintf( stderr, "iphdlr: tcp checksum was correct\n");
                }

                if( cktcpskt( bufferblkp) != 0)
                {       /* is there a socket? */

                        fprintf( stderr, "iphdlr: received unwanted tcp packet\n");
                        bufferblkp->usedflag = 0;       /* no, release the buffer */
                }
                else
                {
                        if( ( status = addbuf2q( bufferblkp, &tcpqptr)) < 0)
                        {
                                /* can we add it to the queue? */

                                bufferblkp->usedflag = 0;       /* no, release the buffer */
                                return( ERROR);                 /* return error */
                        }

                        if( verbose == 259)
                                fprintf( stderr, "ipdhlr: queued TCP packet\n");
                }
        }
        else
        {
                if( iphdrp->proto == IP_PROTO_ICMP)
                { /* is it an ICMP packet? */

                        status = icmphdlr( bufferblkp);     /* yes, process it */
                        if( verbose == 259)
                                fprintf( stderr, "iphdlr: processed ICMP packet\n");
                        if( status  < 0)
                                return( ERROR);
                }
                else
                {               /* else release the buffer */
                        bufferblkp->usedflag = 0;

                        fprintf( stderr, "iphdlr: got non-TCP/UDP/ICMP TCP packet\n");

                        return( 0);
                }
        }
    }
    return( 0);
}

// END IPHDLR.C
```

```
/*---- sendudppkt() ----------------------------------------*/
/* Send 'buflen' number of bytes starting at 'buffer' in a UDP
packet to the host associated with network descriptor 'nd' with
'flags' flags set. Return 'buflen' on success; otherwise return
ERROR.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"
#include   <stdio.h>              /* for NULL, fprintf(), etc. */
#include   <stdlib.h>             /* for rand() */

int sendudppkt( int nd, unsigned char *buffer, unsigned int buflen, unsigned int flags)
{
        extern int neterrno;
        extern int netsuberrno;
        extern struct sockdata *socketroot;
        int net_writeto( int, char *, unsigned int, struct addr *, unsigned int);
        void pneterror( char *);
        int rand( void);
        struct sockdata *sockdatap;
        struct addr *sockinfop;
        int status;

        if( socketroot == NULL)
        {                       /* any open UDP sockets? */

                fprintf( stderr, "sendudppkt: no open sockets to send on\n");
                return( ERROR - 1);                     /* no, error */
        }

        sockdatap = socketroot;                         /* point at the chain */

        while( ( sockdatap->nd != nd) && ( sockdatap->nxtstrcp != NULL))
                sockdatap = sockdatap->nxtstrcp; /* move down the chain until the right connection info is found */

        if( sockdatap->nd != nd)
        {                       /* found the right info? */

                fprintf( stderr, "sendudppkt: can't find structure for nd %d\n", nd);
                return( ERROR - 2);                     /* no, error */
        }

        sockinfop = ( struct addr *) &( sockdatap->fhost);      /* point at the socket info */
        if( sockinfop->lsocket == 0)
        {
                if( sockinfop->fsocket == GETREQPORT)
                        sockinfop->lsocket = sockinfop->fsocket;
                else
                        sockinfop->lsocket = rand();
        }
        if( ( status = net_writeto( nd, buffer, buflen, sockinfop, flags))< 0)
        {
                fprintf( stderr, "sendudppkt: error %d on net_writeto()\n", neterrno);
                if( ( neterrno == NET_ERR_HOST_UNREACHABLE) || ( neterrno = NET_ERR_ICMPMSG))
                        fprintf( stderr, "\tnetsuberrno = %d\n", netsuberrno);

                pneterror( "sendudppkt:");
        }
        return( status);
}
// END SDUDPPKT.C
```

```
/*---- udplisten() ----------------------------------------*/
/* Given given a structure which includes a host number and a socket
number on that host, a socket on this host, and a packet buffer,
prepare to listen for UDP packets from that host to this one.
Return a network descriptor upon successfully initiating listening,
or ERROR on error.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   "\gary\snmp\snmp_src\pd\ipmgr\ipmgr.h"
#include   <stdio.h>                    /* for NULL, etc. */
#include   <malloc.h>                   /* for malloc(), etc. */


int udplisten( long fhost, unsigned int fsocket, unsigned int lsocket, unsigned long option)
{
        extern int neterrno;
        extern int netsuberrno;
        void pneterror( char *);
        int net_getdesc( void);
        int net_listen( int, int, struct addr *);
        int set_option( int, int, int, char far *, int);
        long atonetaddr( char *);
        void *malloc( unsigned int);
        int udprelease( int);
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap;
        struct addr *sockinfo;
        int status;

        if( ( status = net_getdesc()) < 0)
        {       /* got a network descriptor? */

                fprintf( stderr, "udplisten: net_getdesc() returned %d\n", status);
                pneterror( "udplisten: ");
                return( ERROR);                         /* no, error */
        }

        if( socketroot != NULL)
        {                       /* structures in the chain? */
                sockdatap = socketroot;                 /* right, find the end of the chain */
                while( sockdatap->nxtstrcp != NULL)
                {
                        if( ( sockdatap->fhost != fhost) || ( sockdatap->fsocket != fsocket) ||
                                                                ( sockdatap->lsocket != lsocket))
                                sockdatap = sockdatap->nxtstrcp;

                }

                if( ( sockdatap->fhost == fhost) && ( sockdatap->fsocket == fsocket) && ( sockdatap->lsocket == lsocket))
                {
                                /* found an existing structure? */
                        fprintf( stderr, "udplisten: found an existing structure, error\n");
                        return( ERROR - 1);             /* yes, error */
                }
                        /* get memory for a new structure */
                if( ( sockdatap->nxtstrcp = ( struct sockdata *)malloc( sizeof( struct sockdata)))== NULL)
                {
                        fprintf( stderr, "udplisten: malloc error for sockdata to add to chain\n");
                        return( ERROR - 2);
                }
                sockdatap = sockdatap->nxtstrcp; /* add it to the chain */
        }
        else
        {               /* no structures in the chain, make one */
                if( ( socketroot = sockdatap =( struct sockdata *)malloc( sizeof( struct sockdata))) == NULL)
                {
                        fprintf( stderr,"udplisten: malloc error for first sockdata in chain\n");
                        return( ERROR - 3);
                }
```

```
                        /* add it to the chain, too */
        }

        sockdatap->nxtstrcp = NULL;             /* initialize its pointer */
        sockdatap->reqid = ( long)( -2);        /* initialize the request id */
        sockdatap->nd = status;                 /* assign the descriptor */
        sockdatap->routinfop = NULL;            /* traps don't have router info */
        sockdatap->netoption = option;          /* set options in force */
        sockinfo = ( struct addr *) &( sockdatap->fhost);       /* point to the socket info */
        sockinfo->fhost = fhost;                /* set remote host address */
        sockinfo->fsocket = fsocket;            /* set remote host socket */

        sockinfo->lsocket = lsocket;            /* set local socket number */

        if( option == TIMEOUTFL)
        {               /* timeout specified? */
                        /* yes, set timeout value in millisecs */
                if( ( status = set_option( sockdatap->nd, ( int)DGRAM,   ( int)NET_OPT_TIMEOUT, ( char far *)option,
                                                                                        ( int) 0)) < 0)
                {
                        fprintf( stderr, "udplisten: set_option error on timeout, error %d\n",  status);
                        pneterror( "udplisten: ");
                        return( ERROR - 4);
                }
        }
        else if( option == ( unsigned long)NOWAITINGFL)
        {
                                        /* non-blocking option specified? */
                if( ( status = set_option( sockdatap->nd, ( int)DGRAM,
                        ( int)NET_OPT_NONBLOCKING, ( char far *)option, ( int)0)) < 0)
                {
                        fprintf( stderr, "udplisten: set_option error %d\n", status);
                        fprintf( stderr, "udplisten: set_option error on no wait, error %d\n",    status);
                        pneterror( "udplisten: ");
                        return( ERROR - 5);
                }
        }
                        /* try to start the UDP server */
        if( ( status = net_listen( sockdatap->nd, ( int)DGRAM, sockinfo)) < 0)
        { /* error? */
                fprintf( stderr, "udplisten: net_listen returned %d, neterrno = %d\n", status, neterrno);
                if( ( neterrno == NET_ERR_HOST_UNREACHABLE) || ( neterrno = NET_ERR_ICMPMSG))
                        fprintf( stderr, "\tnetsuberrno = %d\n", netsuberrno);

                pneterror( "udplisten: ");
                udprelease( sockdatap->nd);     /* yes, release the allocated memory */

                return( status);
        }
        return( sockdatap->nd);
}

// END UDPLISTN.C
```

```
/*---- udprelease() ----------------------------------------*/
/*  Given a network descriptor, release it.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <malloc.h>                    /* for malloc(), etc. */
#include   <stdio.h>                     /* for NULL, etc. */

int udprelease( int nd)
{
        extern int verbose;
        int net_release( int);
        void free( void *);
        void pneterror( char *);
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap1, *sockdatap2;
        int status;

        if( verbose == 260)
                fprintf( stderr, "udprelease: passed nd = %d\n", nd);

        if( socketroot == NULL)
        {       /* are there any descriptors allocated? */

                fprintf( stderr, "udprelease: no nd's to be released\n");
                if( net_release( nd) < 0)
                        fprintf( stderr, "udprelease: couldn't release nd %d\n", nd);

                return( 0);                     /* no */
        }

        sockdatap2 = sockdatap1 = socketroot;       /* point at the first struct */
        while( ( sockdatap1->nd != nd) && ( sockdatap1->nxtstrcp != NULL))
        {
                /* while this isn't the right structure and more exist, move on to the next structure */

                if( verbose == 260)
                        fprintf( stderr, "udprelease: found nd %d\n", sockdatap1->nd);

                sockdatap2 = sockdatap1; /* save the location of the current struct */
                sockdatap1 = sockdatap1->nxtstrcp;          /* point at the next struct */
        }                                                   /* loop while we haven't found the right struct */

        if( sockdatap1->nd != nd)
        {                       /* did we find a structure? */

                if( verbose == 260)
                fprintf( stderr, "udprelease: didn't find nd %d\n", nd);
                net_release( nd);
                return( 0);                     /* no */
        }                               /* found the right structure */

        if( verbose == 260)
        fprintf( stderr, "udprelease: releasing nd = %d\n", nd);
        if( sockdatap1->nxtstrcp == NULL)
        {                       /* is this the last struct in the chain? */
                if( sockdatap2 == sockdatap1)     /* yes, is it also the first? */
                        socketroot = NULL;              /* yes, clear the root pointer */
                else                                    /* not the first struct */
                        sockdatap2->nxtstrcp = NULL;    /* clear the pointer in the
                previous structure */
        }
        else
        {                       /* this isn't the last structure in the chain */
                if( sockdatap2 == sockdatap1)                   /* is it the first? */
                        socketroot = sockdatap1->nxtstrcp;      /* yes, clear the chain */
                else                                            /* it isn't the first */
                        sockdatap2->nxtstrcp = sockdatap1->nxtstrcp;        /* clear the pointer in the previous structure */
        }
```

```
        free( ( void *)sockdatap1);                     /* free the structure's space */

        if( verbose == 260)
        {
                fprintf( stderr, "udprelease: freed a sockdata blk at %04x\n", sockdatap1);
                fprintf( stderr, "udprelease: socketroot = %04x\n", socketroot);
        }

        if( ( status = net_release( nd)) < 0)
        {       /* release the net descriptor */

                fprintf( stderr,"udprelease: can't net_release() nd %d, status was %d\n",nd, status);
                pneterror( "udprelease: ");
        }

        return( status);
}

// END UDPREL.C
```

```
/*----- udpreleaseall() ----------------------------------------*/
/* Release all network descriptors.

*/

#include   "\gary\snmp\snmp_src\pd\pdnet.h"
#include   <stdio.h>                    /* for NULL, etc. */
#include   <malloc.h>                   /* for free(), etc. */

int udpreleaseall( void)
{
        int net_releaseall( void);
        void free( void *);
        extern struct sockdata *socketroot;
        struct sockdata *sockdatap1, *sockdatap2;

        net_releaseall();

        if( socketroot != NULL)
        {       /* are there any descriptors allocated? */
                sockdatap2 = sockdatap1 = socketroot;          /* point at the first struct */
                while( sockdatap2 != NULL)
                {
                        sockdatap2 = sockdatap1->nxtstrcp;
                        free( ( void *)sockdatap1);            /* free the structure's space */

                        sockdatap1 = sockdatap2;               /* point to the next structure */
                }
                socketroot = NULL;
        }
        return( 0);
}

// END UDPRELAL.C
```

```
/*---- clrvarchgfl() ----------------------------------------*/
/* Given a pointer to a variable info block, clear the value of the
changeflag.  A zero indicates success, negative values indicate
errors.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"        /* for structures */
#include   <stdio.h>                                /* for NULL, fprintf(), etc. */

#define    ERROR              (-1)

int clrvarchfl( variable)
struct varinfo *variable;
{
        extern struct routinfo *firstrinfop;
        struct routinfo *rinfop;
        struct varinfo *vblkp;
        int endflag;

        endflag = 0;
        rinfop = firstrinfop;                              /* make sure 'variable' is a valid pointer */
        while( ( rinfop != NULL) && ( endflag == 0))
        {       /* loop while there are router blocks to look at */

                vblkp = rinfop->vblockp;                   /* point to the first variable block */
                while( ( vblkp != NULL) && ( endflag == 0))
                {       /* loop while there are variable blocks to look at */

                        if( vblkp == variable)             /* is this the right variable block? */
                                endflag++;                 /* yes, set the flag */
                        else
                                vblkp = vblkp->nxtblock;           /* no, continue looping */
                }
                rinfop = rinfop->nxtblock;                 /* go to the next router block */
        }

        if( vblkp != variable)                             /* did we find the right variable block? */
                return( ERROR);                            /* no, error */

        vblkp->changeflag = 0;
                return( 0);
}

// END CLRVARFL.C
```

```
/*---- deldata() ----------------------------------------*/
/* Given a pointer to a string of four hexadecimal bytes containing
an Internet address, invalidate the data in the SNMP database
associated with the address. Return zero on success, a negative
number on failure.

*/

#include   "\gary\snmp\snmp_src\snmp\snmp.h"        /* get SNMP variables */
#include   <stdio.h>                                /* for fprintf(), etc. */
#include   <sys/types.h>                            /* for 'ftime()' */
#include   <sys/timeb.h>                            /* for 'ftime()' */

#define    ERROR              (-1)

int deldata( unsigned char *ipaddr)
{
        struct routinfo *getrouterp( char *, struct routinfo *);
        struct varinfo *getnxtvarp( struct routinfo *, struct varinfo *);
        void ftime( struct timeb *);
        struct routinfo *rinfop;
        struct varinfo *vinfop;
        int i, j;
        char ipaddrs[18];
        struct timeb  timeinfo;

        for( i = 0, j = 0; i < 4; i++)
        {       /* convert the four hex bytes to a null-terminated ASCII string */

                j += sprintf( ( &ipaddrs[ j]), "%u.", ( unsigned int) ipaddr[ i]);
        }

        ipaddrs[ --j] = '\0';

        rinfop = NULL;                  /* look for the first agent info block */
        vinfop = NULL;                  /* look for the first variable info block */
        while( ( rinfop = getrouterp( ipaddrs, rinfop)) != NULL)
        {       /* while there are agent blocks remaining with the right IP address, void the data in the variable blocks */

                while( ( vinfop = getnxtvarp( rinfop, vinfop)) != NULL)
                {
                                        /* while there is a variable block, */
                        switch ( vinfop->vartype)
                        {       /* find the type of variable */

                                case ( UINTID):                 /* its an integer value */
                                        *( ( int *) ( vinfop->varvalue)) = -1;
                                        break;

                                case ( UOCTSTRID): /* string value */
                                case ( DisplStrID):     /* display string */
                                case ( IntegerStrID):   /* integer string */
                                        *( ( int *) ( vinfop->varvalue)) = 1;
                                        *( ( unsigned char *)
                                         ( ( ( int *) ( vinfop->varvalue)) + 1)) = '\0';
                                        break;
                                case ( UOBJIDID):       /* object identifier */
                                        *( ( int *) ( vinfop->varvalue)) = 1;
                                        *( ( unsigned char *)
                                        ( ( ( int *) ( vinfop->varvalue)) + 1)) = '\0';
                                        break;
                                case ( IpAddrID):       /* Internet address value */
                                        for( i = 0 ; i < 4 ; i++)
                                        {
                                                ( ( char *) ( vinfop->varvalue))[ i] = '\0';
                                        }
                                        break;
                                case ( CounterID):   /* counter value */
                                case ( GaugeID):                /* gauge value */
                                case ( TimeTicksID):            /* time ticks value */
                                        *( ( long *) ( vinfop->varvalue)) = ( long)( -1);
```

```
                                                    break;
                                }
                                vinfop->changeflag++;
                                ftime( &timeinfo);                  /* get the time of the change */
                                vinfop->timestamp = timeinfo.time;          /* save it */
                        }
                }
                return( 0);
}

// END DELDATA.C

/*----- endrcv() ----------------------------------------------*/
/* Clean up before stopping the receipt of SNMP packets.  Revised

*/

int endrcv()
{
   int udpreleaseall();
   udpreleaseall();
   return( 0);
}


// END ENDRCV.C

/*----- endsnmp() ---------------------------------------------*/
/*  Clean up all loose ends and turn off the UDP packet driver.

*/

int endsnmp( void)
{
   int endudp( void);
   endudp();
   return( 0);
}


// END ENDSNMP.C
```

Appendix C

```
/*---- objects' database ----------------------------------------*/
/* The main program Reads a file to determine what agents to interrogate
    and which variables to monitor.  The file has the format

<Configuration>                                 # configuration information follows
        pktdrvr                                 # packet driver configuration information
                locipaddr=x.x.x.x               # device Internet address
                ipmask=x.x.x.x                  # Internet network mask
                gw=x.x.x.x                      # primary gateway's address

community
        agent_address           # comment
                variable_name           index_value             # comment
                variable_name           index_value             # comment
                (this is a blank line - whitespace, e.g. '\t', ' ', '\n')
        agent_address           # comment
                variable_name           index_value             # comment
                variable_name           index_value             # comment
                        (this is a blank line)
        agent_address           # comment
                variable_name           index_value             # comment
                variable_name           index_value             # comment
                (this is a blank line or end of file)
```

White space can be spaces or tabs.

The example in the following page is a copy actual text file used to develope the SNMP management tool.

Example; used for UTC management tool application.

```
<Configuration>                                  # configuration information follows
        pktdrvr                                  # packet driver configuration information
                locipaddr=192.239.44.47          # device Internet address
                ipmask=255.255.255.0             # Internet network mask
                gw=192.239.44.1                  # primary gateway's address

public                                           # 1st SNMP community
    192.239.44.1                                 # first device in this commun.
        ifOperStatus        "1"                  #get operational status of interface 1
        ifOperStatus         2                   # get same of interface 2
        ifOperStatus        "3"                  # ditto for interface 3
        sysObjectID                              # value = "43.6.1.4.1.1.1.1.42"

    192.239.44.1
        sysUpTime                                # value = 52274315
        ifType              1                    # value = "ethernet-csmacd"
        ifType              2                    # value = "proPointToPointSerial"
        ifType              3                    # value = "ethernet-csmacd"

    192.239.44.1
        ifIndex             1                    # value = 1
        Descr               1                    # value = Ethernet

    192.239.44.1
        ifInErrors          1                    # value = 95
        ifOutErrors         1                    # value = 1243
        ifInErrors          2                    # value = 79636
        OutErrors           2                    # value = 10

    192.239.44.1
        ifInErrors          3                    # value = 3265
        ifOutErrors         3                    # value = 37
        ifMtu               1                    # value = 1500
        ifSpeed             1                    # value = 10000000

    192.239.44.1
        ifMtu               2                    # value = 1500
        ifSpeed             2                    # value = 448000
        ifMtu               3                    # value = 1500
        ifSpeed             3                    # value = 10000000

    192.239.44.1
        ifPhysAddress       1                    # value = 00-00-0c-02-48-be
        ifAdminStatus       1                    # value = "up"
        ifPhysAddress       2                    # value = 00-00-d3-71-87-c4
        ifAdminStatus       2                    # value = "up"

    192.239.44.1
        ifPhysAddress       3                    # value = 00-00-f2-82-90-a1
        ifAdminStatus       3                    # value = "up"
        ifInUcastPkts       1                    # value = 31900562
        ifInUcastPkts       2                    # value = 20603518

    192.239.44.1
        ifInUcastPkts       3                    # value = 39683085
        ifInNUcastPkts      1                    # value = 132347
        ifInDiscard         1                    # value = 0
        ifInNUcastPkts      2                    # value = 923170

    192.239.44.1
        ifInDiscard         2                    # value = 0
        ifInNUcastPkts      3                    # value = 1122039
        ifInDiscard         3                    # value = 0
        ifInErrors          1                    # value  = 95
```

```
192.239.44.1
        ifInUnknownProtos   1                   # value = 0
        ifInErrors          2                   # value  = 79636
        ifInUnknownProtos   2                   # value = 0
        ifInErrors          3                   # value  = 3265

    192.239.44.1
        ifInUnknownProtos   3                   # value = 0
        ifOutOctets         1                   # value = 32904802
        ifOutUcastPkts      1                   # value = 31709958
        ifOutOctets         2                   # value = 20838652
        ifOutUcastPkts      2                   # value = 46161389

    192.239.44.1
        ifOutOctets         3                   # value = 19714933
        ifOutUcastPkts      3                   # value = 33709958
        ifOutNUcastPkts     1                   # value = 9714933
        ifOutDiscard        1                   # value = 30

    192.239.44.1
        ifOutNUcastPkts     2                   # value = 0
        ifOutDiscard        2                   # value = 571356
        ifOutNUcastPkts     3                   # value = 8685823
        ifOutDiscard        3                   # value = 37

    192.239.44.1
        ipDefaultTTL                            # value = 255
        ipInReceives                            # value = 74367593
        ipInHdrErrors                           # value = 156
        ipInAddrErrors                          # value = 0

    192.239.44.1
        ipForwDatagrams                         # value = 73164027
        ipInUnknownProtos                       # value = 0
        ipInDscards                             # value = 0
        ipInDelivers                            # value = 614769

    192.239.44.1
        ipOutRequests                           # value = 1233031
        ipOutDiscards                           # value = 10872
        ipOutNoRoutes                           # value = 6728
        ipReasmTimeout                          # value = 30

    192.239.44.1
        ipReasmReqds                            # value = 0
        ipReasmOKs                              # value = 0
        ipReasmFails                            # value = 0
        ipFragOKs                               # value = 392

    192.239.44.1
        ipFragFails                             # value = 0
        ipFragCreates                           # value = 0
        ipAdEntIfIndex      192.239.44.1        # value = 3
        ipAdEntNetMask      192.239.44.1        # value = 255.255.255.0
```

Appendix D

User's Guide

Introduction

Welcome to SNMPMGR, a network management tool to help you to monitor the network devices using Simple Network Management Protocol. SNMPMGR is a vendor independent tool that runs on very low cost hardware, yet provides the basic functions one needs to monitor the health of network devices such as routers and gateways.

Hardware and Software Requirements

SNMPMGR runs on the IBM PC family of computers, including the XT, AT, and PS/2, along with all true compatibles. SNMPMGR uses the packet driver to communicate on the Ethernet network. Therefore, the computer on which the SNMPMGR is going to be running needs an Ethernet interface card with the packet driver for the card. SNMPMGR requires MS-DOS 2.0 or later, and at least 256KB of memory.

Installing SNMPMGR

Before starting to use the SNMPMGR program, you will need to get one thing out of the way. Take the SNMPMGR disk and make a copy (via DOS) of the disk for your working copy. Once you have done that, put the original disk away. SNMPMGR can be run from a floppy disk or a hard disk. If you would like to run the program from a hard disk, copy the SNMPMGR disk to your hard drive using the DOS copy command. The disk contains three files, SNMPMGR.EXE, UTC.DB (UTC network specific) and MIB.2. In order to run the program you need to have these files in your current directory.

Getting SNMPMGR Running

To start the program type SNMPMGR with a command line parameter specifying the exact objects to be accessed in each network device (the SNMP objects configuration file). The following example provides the file UTC.DB as the objects configuration file and starts the program from the C drive..

C> SNMPMGR UTC.DB

The objects configuration file which describes what devices and objects are to be accessed has a required format. This format allows the program to separate the SNMP Community string and the device Internet address from the SNMP Objects in the device which will be accessed. Some knowledge of SNMP may be required to properly specify a device object. The file UTC.DB is provided as an example of an objects configuration file. If the objects configuration file does not have the correct format, the program will exit with an error message that " Error in objects configuration file". Therefore, it is important having correct objects configuration file as described in the following section.

The Objects Configuration File Format

Community and Object Lists

A community list is a community string, which is followed by one or more object lists. Community lists are terminated by the end of the objects configuration file, or by encountering another community string. Object lists within a community list are separated by blank lines. Lines containing comments are not considered blank lines.

Each object list must be part of a community list. Therefore, the first non-comment and non-blank line must contain a community string. After a community has been specified, that name is used as the default community string for following object

lists until a new community string is encountered. When a new community string is encountered, it becomes the default community string. Any object lists not immediately following a community string are given the default community string. Community strings have unique properties so that community lists are separated from each other.

An object list consists of the agent's name followed by the objects and associated indices to be accessed in that agent. The index for an object must be on the same line as the object and separated from it only by one or more spaces or tab characters. No blank lines may appear in an object list.

Community strings

A community string must start at the beginning of a line, and must not begin with the '#' character or ' " ' character. Should it do so, it will be considered a comment, not a community string. No other names in the SNMP database file start at the beginning of a line. A community string is terminated by a space, tab, or end-of line character, unless the name is enclosed in double quote characters. When a community string begins with a double quote character, i.e., the double quote character is the first character on the line, the community string is terminated by another double quote character and so may include space or tab characters. The double quote characters enclosing a name are not considered part of the name when SNMP messages are generated.

Agent names

Since agent names must not start in the first character position on a line, they must be preceded by one or more spaces or tab characters. If the agent name immediately follows a community string, the agent name may be on the same line as the community string, but separated from it by one or more spaces or tabs. If a community string is not supplied immediately before an agent name, the application will assign the default community string to that agent. An agent name is terminated by a space, tab, or

end-of-line character. The Agent names can be only the agent's Internet address in standard Internet "dot" notation (such as 192.239.44.1). Since no spaces or tabs are allowed in an agent name, it need not be enclosed in double quotes. If it is enclosed in double quotes, they will be ignored.

Object names

Like agent names, object names may not begin with the first character on a line and are terminated by a space, tab, or end-of line character. Valid object names were defined in Chapters 2-4, and must be entered exactly in the form presented there, including spelling and capitalization. Object names need not be enclosed in double quote characters because no spaces are allowed in them.

Object names must not be separated from each other, or the agent name they are associate with, by blank lines. Blank lines separate object lists from each other.

Indices

An index must be on the same line as the object it modifies. It must be separated from the object name by one or more spaces or tabs. If an object name has no index on the same line, the application assumes no index is to be associated with that object.

An index is specific to its associated object. Some objects do not require indices; others require multiple indices, such as interface objects. When multiple indices are required, they are separated by either a single space or comma. When they are separated by a space, the entire set of indices must be enclosed by a pair of double quote characters. When double quotes are not used, the index name is terminated by the first space, tab, or end-of-line character encountered.

Comments

A comment is initiated by a '#' character either at the beginning of a line in the objects configuration file, or elsewhere in a line preceded by a space or tab character. A comment is terminated by the end of the line on which it was started. A '#' character not at the beginning of a line and not preceded by a space or tab character will not be considered as the beginning of a comment, but will be included in the field currently being scanned.

Local Configuration information

The only network interface configuration information required to run the application is that for a packet driver. Three objects must be provided: the local interface's IP address, the IP network mask, and the local gateway's IP address. All three values are assigned by the manager of the network on which the host running this application resides.

Local Configuration information is entered in the objects configuration file in a format similar to that of a community list. The community string for the local configuration information must be '`<Configuration>`' ("<>" brackets included). The agent field must be 'pktdvr'. The objects are '`locipaddr=x.x.x.x`' for the local IP address, '`ipmask=x.x.x.x`' for the IP network mask, and '`gw=x.x.x.x`' for the local gateway's IP address. In the examples each 'x' is an integer between 0 and 255, inclusive. If the network is not using any subnet the value of ipmask would be 255.255.255.0 to indicate there is not any subnet for the network.

Displaying Objects

When the program starts, it reads the supplied objects configuration file. If the objects configuration file has the correct format, it will interrogate the specified agents for their values of the indicated objects. When a packet is received from an agent, the

values of the objects will be printed on the console with the agent name (IP address of the agent) and a time stamp. Then the program goes inside a loop (main loop) to repeat the above process until the exit key is pressed (**CTRL-END**) as shown in figure 6.1.

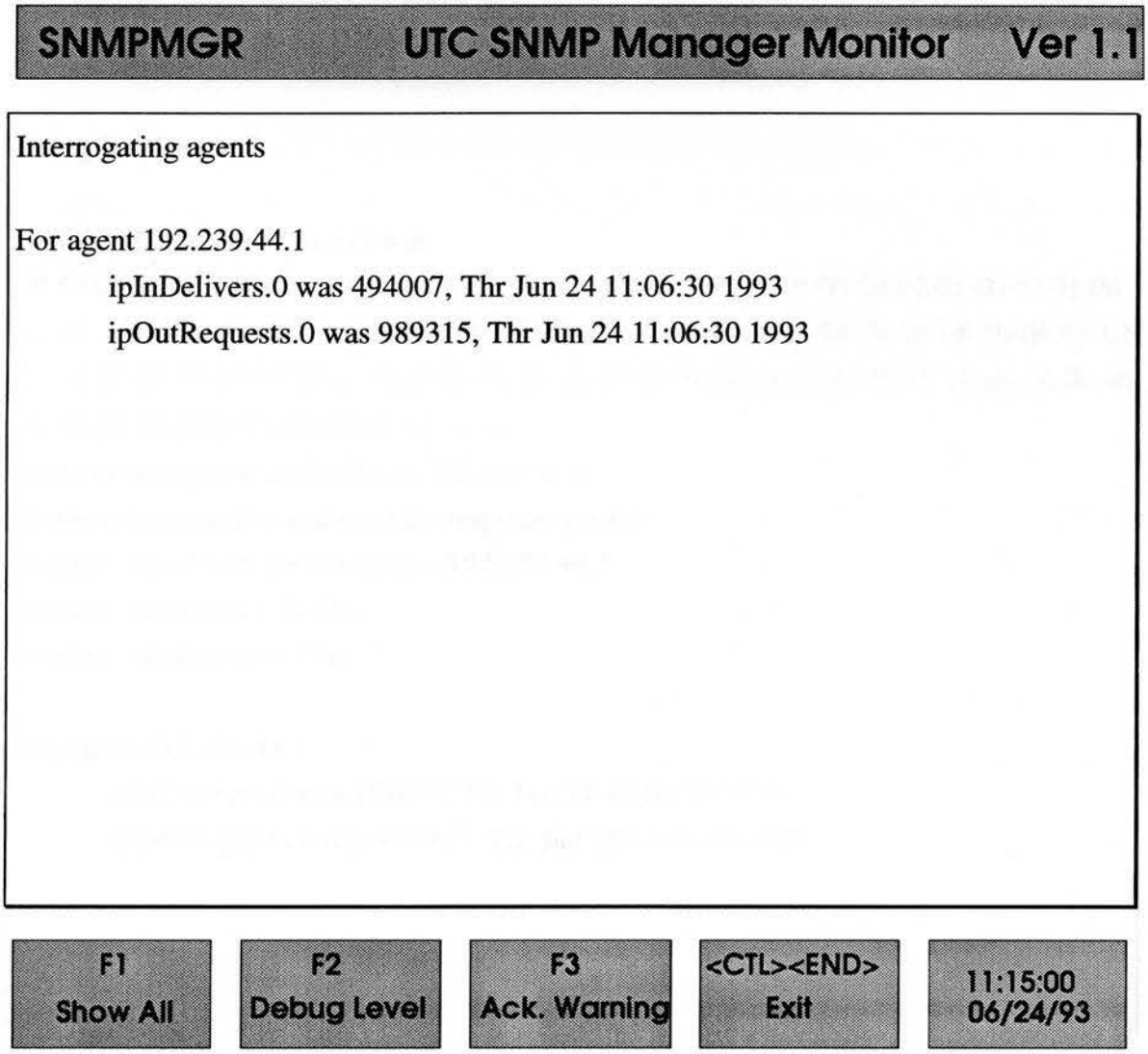


Figure D.1. Example Display Screen.

If function key 1 (**F1**) has been pressed, the program will display the value of the objects on the screen. If the function key 2 (**F2**) has been pressed, the debug level is entered which is either 1 or 2. Debug level 1 causes the content of the next packet to be

printed in hexadecimal. Debug level 2 causes the content of the next packet along with the connection information to be printed in hexadecimal as shown in Figure 6.2.

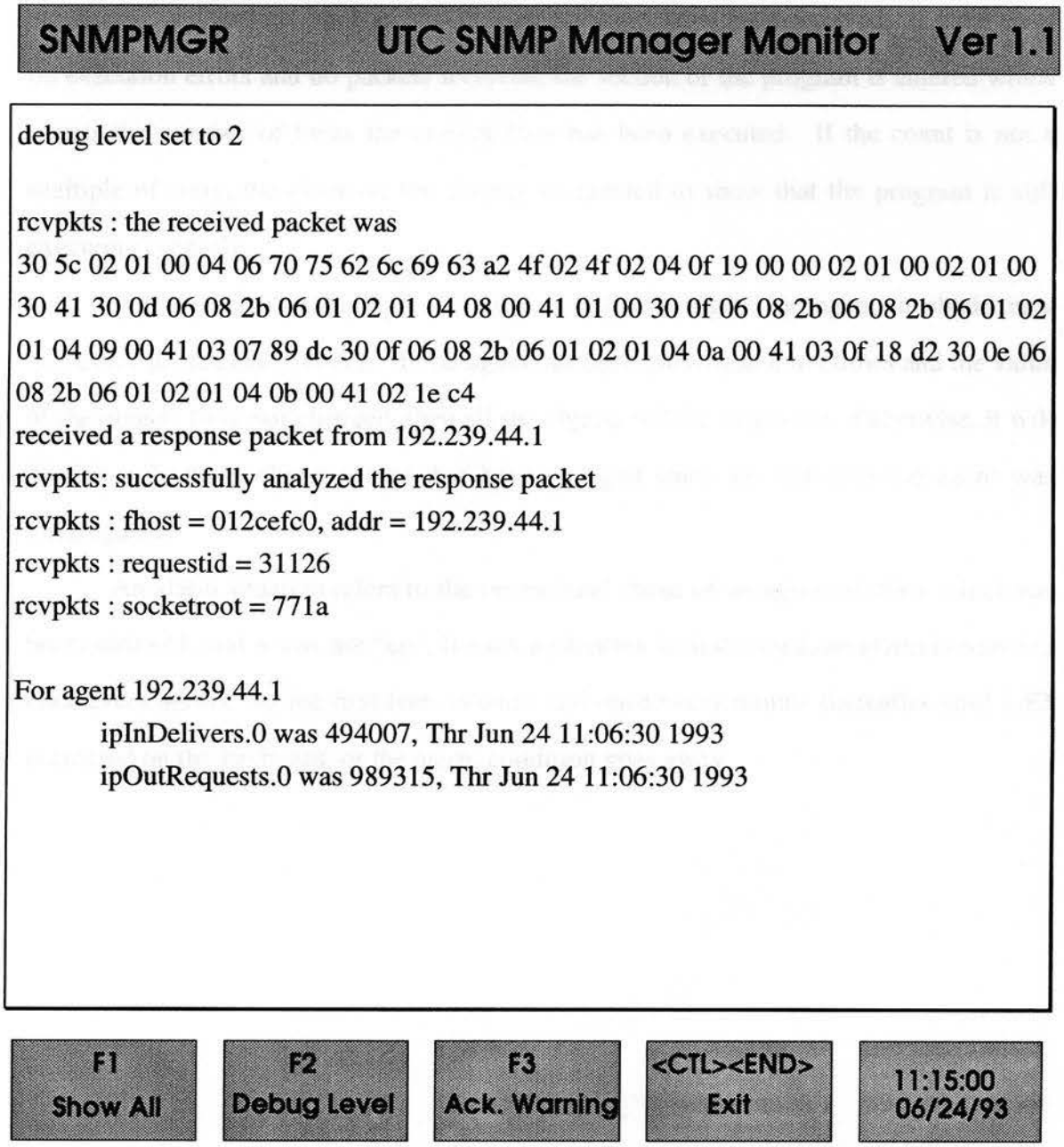


Figure D.2. Example Debug Screen.

When function key 3 (**F3**) has been pressed, any alarms that might be sounding (described below) are disabled. If **CTRL-END** (EXIT) has been hit, it will notify the program to exit gracefully. If no key has been pressed the program will check to see if any unsolicited packets, such as SNMP trap messages, have been received. If there were no execution errors and no packets received, the section of the program is entered which counts the number of times the current loop has been executed. If the count is not a multiple of sixty, the clock on the display is updated to show that the program is still executing properly.

If the count is a multiple of sixty, it will interrogate the agent to obtain new values for the database objects. If the agent has been interrogated ten times and the value of the objects have not changed, then all the objects will be displayed. Otherwise, it will display only those object values that have changed since the last time the agent was interrogated.

An alarm situation refers to the operational status of an agent interface which has been retrieved, and it was not "up". If such a situation is discovered, an alarm is sounded once every second for the first five seconds, and once every minute thereafter until a **F3** is entered on the keyboard, or the alarm condition goes away.