1997

Hardware Interfacing in the Broadcast Industry Using Simple Network Management Protocol (SNMP)

Walter H. Schuller Jr.
University of North Florida

Suggested Citation

Schuller, Walter H. Jr., "Hardware Interfacing in the Broadcast Industry Using Simple Network Management Protocol (SNMP)" (1997). *UNF Graduate Theses and Dissertations*. 339.
https://digitalcommons.unf.edu/etd/339

HARDWARE INTERFACING IN THE BROADCAST INDUSTRY USING SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

by

Walter H. Schuller Jr.

A thesis submitted to the
Department of Computer and Information Sciences in partial
fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

August, 1997

The thesis "Hardware Interfacing in the Broadcast Industry using Simple Network Management Protocol (SNMP)" submitted by Walter H. Schuller Jr. in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences at the University of North Florida.

Approved by the thesis committee: Date

Signature Deleted

8/6/97

Dr. Yap S. Chua
Thesis Adviser and Committee Chairman

Signature Deleted

8/12/97

Dr. Ralph Butler

Signature Deleted

8/6/97

Dr. Robert F. Roggio

Accepted for the Department of Computer and Information Sciences:

Signature Deleted

8/8/97

Dr. Charles N. Winton
Chairperson of the Department

Accepted for the College of Computing Sciences and Engineering:

Signature Deleted

8/8/97

Dr. Charles N. Winton
Acting Dean of the College

Accepted for the University:

Signature Deleted

8/10/97

Dr. William J. Wilson
Dean of Graduate Studies

Acknowledgment

I wish to thank my wife and children for their understanding and support during this education experience. I would also like to thank the management of W.J.X.T. TV-4 (my employer) for all their support as well. Thanks to all involved, I am a better person for what I have been allowed to accomplish.

CONTENTS

FIGURES

ABSTRACT

Communication between various broadcast equipment plays a major role in the daily operation of a typical broadcast facility. For example, editing equipment must interface with tape machines, production switchers must interface with font generators and video effect equipment, and satellite ground controllers must interface with satellite dishes and receivers. Communication between these devices may be a simple hardware handshake configuration or a more elaborate software based communications via serial or parallel interfacing. This thesis concerns itself with the software interfacing needed to allow various dissimilar types of equipment to communicate, and therefore, interface with each other. The use of Simple Network Management Protocol (SNMP) in a non-typical manner for the purpose of hardware interfacing is the basis for this work.

Chapter 1

INTRODUCTION

One of the problems continuously facing the broadcast industry is how to interface various types of equipment from different manufacturers. Unfortunately, standardization of hardware interfacing is not as prevalent in the broadcast industry as it is in others. Because of this lack of standardization, manufacturers are forced to write machine specific code when attempting to interface micro-processor based equipment. This software is used for communications between the dissimilar devices over various topologies, and this arrangement often causes a second problem. Each unit must be directly connected with the other unit to which it is interfacing; therefore, physical location becomes an issue. This complicates equipment installation and is an

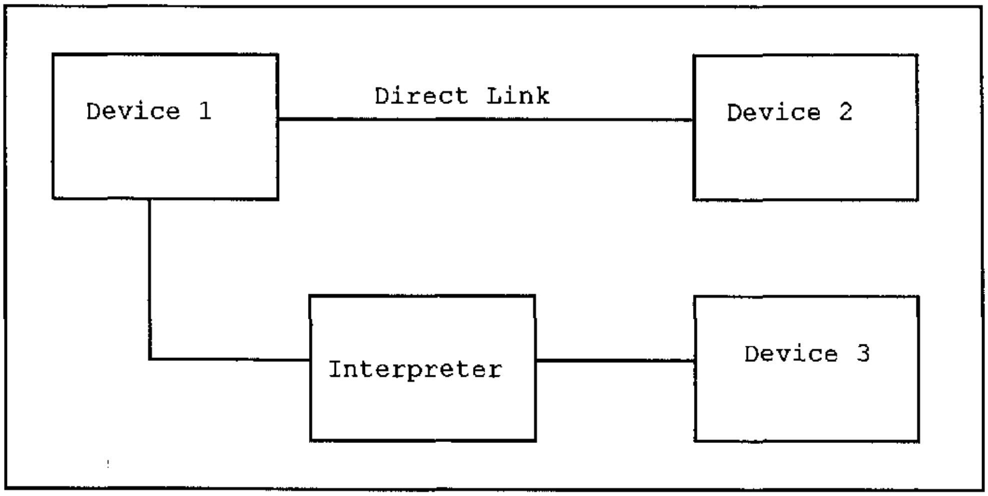


Figure 1: Typical Interface Scheme

inefficient use of expensive signal wire. Another technique is to use a third computer as an interpreter between two products wishing to communicate. This machine receives communication control codes from one device and translates the request into the required command string for the other device. Obviously, this is a very inefficient use of hardware. Figure 1 shows the two typical types of interfacing used in the broadcast industry today.

A more efficient and standard interfacing scheme is needed, and could be easily employed. Local Area Networks (LANs) are becoming common in the broadcast industry, and should be considered as a medium for communication between broadcast equipment. The problem of hardware dependent code for communicating and interfacing could be eliminated by use of a standard network protocol. One such established protocol

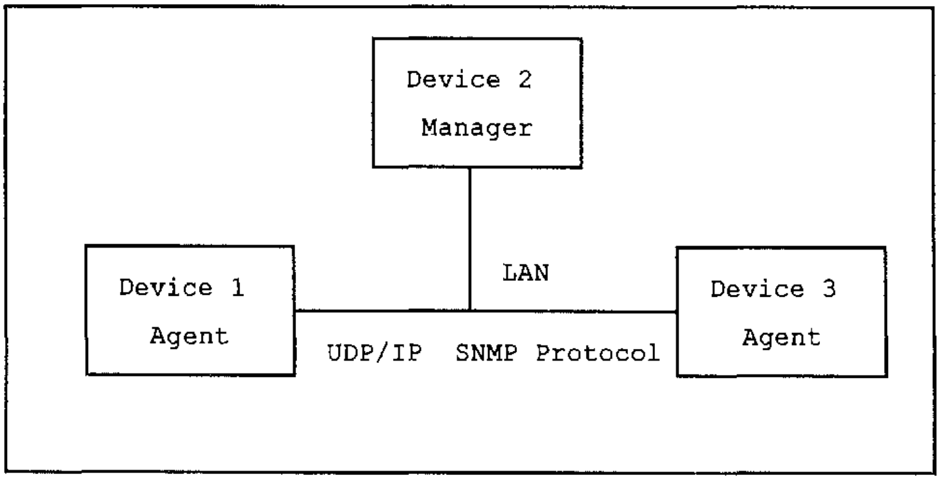


Figure 2: Interfacing via LAN and SNMP

used for network monitoring is the Simple Network Management Protocol (SNMP) of the TCP/IP suite. Although SNMP was not intended for this type of equipment interfacing, it is my belief that SNMP could be used for limited equipment interfacing between different manufacturers; thus, eliminating the need for hardware dependent software. Each manufacturer would simply provide a network interface (such as Ethernet), an SNMP agent server, and a MIB for its product. This established protocol would allow other manufacturers to easily communicate, and thereby interface, with the other vendor's equipment by the use of a SNMP manager (Figure 2). Previously released products could be retrofitted by constructing a network interface with a proxy server for the device.

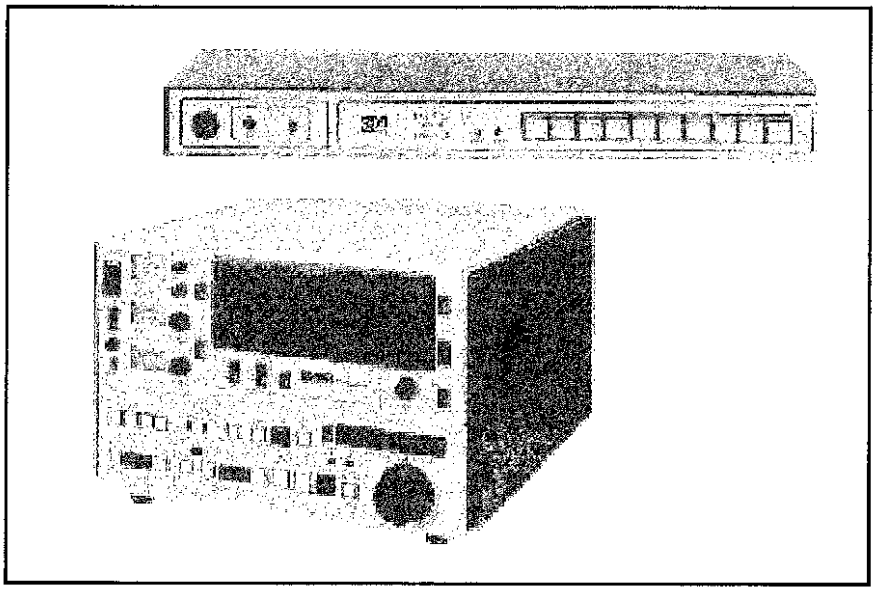


Figure 3: Sony Tape Machine and Audio/Video Switcher

In this work, I shall explore the feasibility of using SNMP as a common interface. With this protocol, I hope to control an audio/video vertical interval switcher (figure 3), a Sony tape machine (BVU-800) (figure 3), and an emergency transmitter controller as examples of my proposal. The transmitter controller was designed and constructed by this student as a senior project while working on an undergraduate degree in electrical engineering. In all three cases, I will use personal computers running Windows 95 as proxy agents and managers (the programs should run equally as well with Windows 3.11). Physical control will be through the serial ports of the computer. This should prove the validity of using SNMP for equipment control as well as the possibility of retrofitting existing equipment. Speed and efficiency of equipment control as well as network traffic problems as a result of this type of interface will

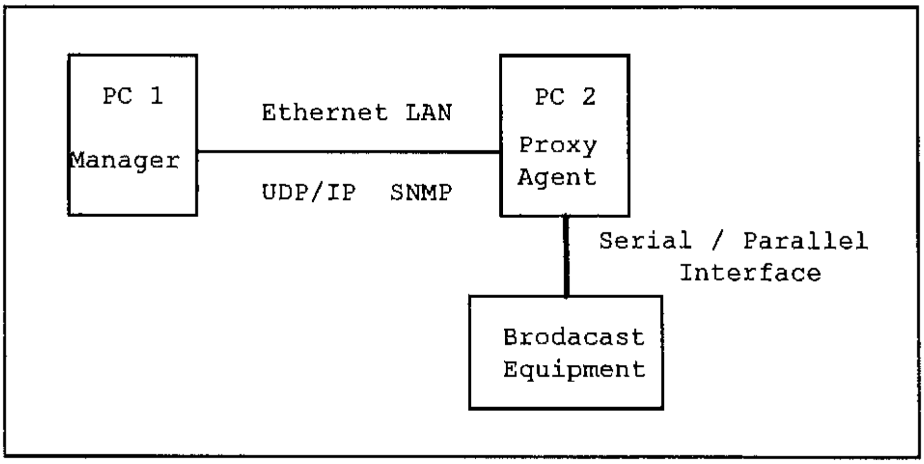


Figure 4: Equipment Interconnection

be studied. This work will also include examples of proposed MIBs for other types of broadcast equipment that could possibly be controlled using SNMP.

It is not the intent of this thesis to educate the reader in all aspects of SNMP or the broadcast environment, nor is it the intent of this thesis to exercise every possible combination of equipment interaction. This thesis only attempts to prove the concept of using the Simple Network Management Protocol to interface dissimilar units of broadcast equipment.

Chapter 2

SNMP - THEORY AND IMPLEMENTATION

The Simple Network Management Protocol (SNMP) is a TCP/IP based protocol used for network management. Network elements (printers, routers, servers, and etc.) can be monitored and/or controlled through this management tool. Communications is based on a client/server arrangement. The client (network manager) communicates with a server (SNMP agent). Together, the manager and agent software maintain a common database called the Management Information Base (MIB) for each controlled or monitored device. Limited control of a device, such as system reset, is performed by changing a MIB variable which causes the agent to act accordingly. Monitoring of an element is accomplished by requesting the agent program to provide MIB data corresponding to the monitored function desired.

It should be noted that this unique protocol can be used for more than just network monitoring. It has also been used to monitor heating and cooling control networks, automotive traffic networks, chemical and industrial processes and many other real-time system applications. Over time, this simple protocol has proven to be quite useful in applications other than its intended use.

This protocol is purposely designed to be small in size in order to keep processor overhead and network traffic as low as possible. The SNMP agent software is usually quite small (often less than 64k). The protocol uses the polling technique and UDP (User Datagram Protocol) to help keep efficiency as high as possible. The simplicity of the connectionless datagrams of UDP helps with debugging as well. " As network debugging in the face of changing routes will certainly mean losing packets, retaining this control from the transport service (layer 4) was considered essential. Since a network management protocol will be run continuously it is mandatory that it consume as minimal network resource as possible. UDP allows the necessary control over packet transmissions, packet size and content (packetization). It was a natural choice" [Satz].

SNMP uses WELL KNOWN port numbers 161 and 162. Management request and agent responses use port 161 while agent trap messages use port 162. Basically, the manager program obtains a port address from its pool of unused ports, and includes this address as the source in the SNMP message being sent to port 161 of the agent program. The agent program uses the source address of the manager for the GET-RESPONSE message. The agent program uses port 162 as its target address of the manager for all TRAP messages.

The SNMP protocol supports five control primitives. Three are used by the manager software while the other two are used by the agent software. The manager uses GET REQUEST to obtain status information of the device from an agent. The agent uses a GET RESPONSE message to provide that information. If the amount of data is too great for one message, then the agent will send what it can and the manager will request the next packet of data using the GET NEXT REQUEST.

MANAGER	GET REQUEST - Used to request MIB data.
MANAGER	GET NEXT REQUEST - Used to get next sequential data unit from MIB.
MANAGER	SET REQUEST - Used to set variables in MIB.
AGENT	GET RESPONSE - Used to respond to GET REQUEST, GET NEXT REQUEST, and SET REQUEST.
AGENT	TRAP - Used to report unsolicited device event information.

Figure 5: The Five SNMP Control Primitives

A manager can also set the values of various object identifiers (variables) which in some cases will cause the agent software to take appropriate action on the device. Finally, an agent using the trap message format can alert the manager to a change in device status such as a system error.

The generated action by the agent server in response to the manipulation of objects within the MIB by the managing client is the basis of this thesis. Setting, or resetting, objects will cause the agent to control various functions of an interfaced device. However, notice that the agent program is doing all the work in this implementation which is opposite to the way the protocol was designed. Under normal operation, SNMP puts more of the work load on the manager program to prevent the agent from robbing system resources from its host device. The work-loaded agent should not be a problem in my implementation due to the nature of the equipment being controlled. Much of the time, the hardware is waiting for the next command. The speed of the controlling processor is considerably faster than the mechanics of the machine or the user operating it.

A couple of other characteristics of the SNMP may be worth noting. All implementations of SNMP must be able to receive messages of at least 484 octets in size; however, UDP packets can be as large as 65k. In most normal implementations of SNMP, the manager polls the agents on regular intervals (typically 15 minutes).

For equipment not able to handle SNMP, another device with an agent package can act as a proxy agent for the unit. For example, an agent running on a workstation can be used as a proxy agent for a printer which is attached to that workstation. Proxy agents are also useful for load reduction on heavily used equipment. The system running the proxy agent will take the task of management off the overworked system. As stated previously, this thesis employs proxy agents for control of the various broadcast devices used.

The Management Information Base (MIB) uses only a few different types of data to describe the status, performance, configuration, and etc. of a device. Each device has its own MIB which is used for its control and status reporting. The fundamental ASN.1 data types used with SNMPv1 are NULL, OCTET STRING, INTEGER, and OBJECT INDENTIFIER. All other data types are derived from these basic units and are described in MIBs which were declared in early RFC's. This practice of deriving new data types is still in practice.

Every variable in a MIB must be referenced by its object identifier name. An object identifier is authoritatively named using a tree structure similar to the DNS of the Internet (figure 6). Most variables start with the numeric name of 1.3.6.1.2.1 which corresponds to a textual name of iso.org.dod.internet.mgmt.mib. Two exceptions to this are vendor-specific MIBs whose variables start with the numeric name of 1.3.6.1.4.1 and the experimental MIBs with the numeric names of 1.3.6.1.3. These identifiers correspond to textual name of iso.org.dod.internet.private.enterprises and iso.org.dod.internet.experiment respectively. This thesis used the experimental identifier prefix, and a partial example of a MIB created for this project is shown in figure 7 (next page). The IMPORT of "experimental" from RFC1155-SMI sets the prefix of the modules to 1.3.6.1.3 (iso.org.dod.internet.experiment).

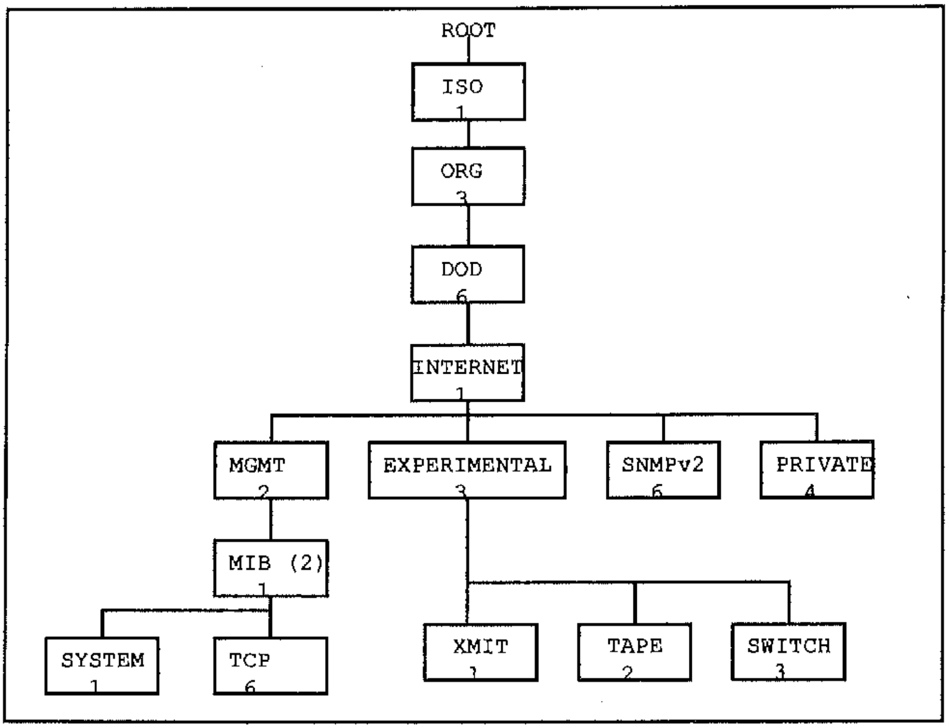


Figure 6: Object Identifier Tree

It should be noted that only node leaves of the MIB name tree may be referenced. Variables are referenced by adding an index to the numeric identifier. A variable name has the following format: OBJECET INDENTIFIER.INDEX. One of a kind objects have a zero for their index. For example, a variable known as udpInDatagrams would be referenced as 1.3.6.1.2.1.7.1.0 which corresponds to the text identifier of iso.org.dod.internet.mgmt.mib.udp.udpInDatagrams.0, and the variable know as txExciterA (following page) would be referenced as 1.3.6.1.3.3.0 which corresponds to iso.org.dod.internet.experiment.txExciterA. These and other detailed examples can be found in Understanding SNMP MIBs by David Perkins and Evan McGinnis.

```
THESIS-MIB DEFINITIONS ::= BEGIN

-- Title:  UNF Thesis MIB Examples
-- Date:   May 1997
-- By:     Walter Schuller <wschul@osprey.unf.edu>

IMPORTS
experimental               FROM RFC1155-SMI
OBJECT-TYPE                FROM RFC-1212;

-- Objects for TRANSMITTER control MIB

txPowerUp OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transmitter power to be
     increased by a predetermined amount."
    ::= { xmit 1 }

txPowerDown OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transmitter power to be
     decreased by a predetermined amount."
    ::= { xmit 2 }

txExciterA OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes exciter A to be activated."
    ::= { xmit 3 }

~~~~~~~~~~~~~~~~~~~~~~

END
```

Figure 7: Sample Experimental MIB

MIBs often need to be checked for correct syntax and/or translated into various useable output formats, such as data structures, which may be directly inserted into a program's source code, or which may be read directly by SNMP management or agent applications at run-time. A special MIB compiler is used to perform these functions. "MIB compilers tend to be greatly misunderstood tools. This term is usually applied to a MIB syntax checker authors use to ensure that their MIBs are written in the correct form, but it also applies to an entire class of tools that perform functions as diverse as drawing tree representations of a MIB, to automatically generating C code for a management application or agent" [325, Perkins]. Basically, MIB compilers are simple translators which work with an adapted sub-set of the ASN.1 description language. Note that an ASN.1 compiler should not be used as a MIB compiler because this type of compiler might not recognize the SNMP adaptations of the ASN.1 language.

MIB compilers are similar in construction to regular compilers. They often require multiple passes during the compiling process (see figure 4). The first phase of compiling is done in the front-end. This is where the syntax checking is performed. Output from this phase is an intermediate code which is fed to the back-end compiler. The output of this phase is determined by which back-end

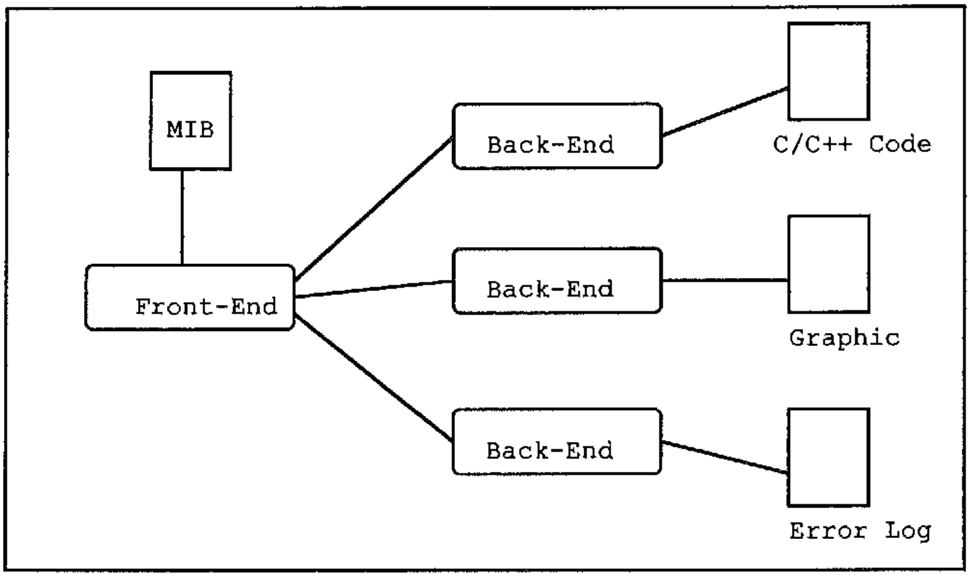


Figure 8: Typical MIB Compiler

compiler is chosen. It could be a graph displaying the MIB overview, a report displaying all errors, source code providing a structure definition for the MIB, or a special format that a manager or agent program could read during run-time.

The MIB compiler used in this thesis was the SNMP Management Information Compiler - Next Generation (SMICng) by SynOptics Communications, Inc. It can be obtained from this company or from a CDROM included with the text <u>Understanding SNMP MIBs</u>. It was mainly used to check for correct syntax of the experimental MIBs included in this work.

The Simple Network Management Protocol is not perfect. Although the SNMP protocol appears to be quite simple (only five primitives), the MIB is somewhat more complicated.

Combined with the BER encoding rules, SNMP is not as simple to implement as the name implies. The protocol has also been criticized for its inefficient use of bandwidth. This, in part, is due to the polling technique, and the transmission of needless information, such as the version number and large data handles with each transmission packet. Another disadvantage is due to the unreliability of the UDP protocol; however, due to its simplicity when compared to TCP, it is also one of its strengths.

Some of these and other problems have been addressed in version 2 of this protocol (SNMPv2). Improvements deal with larger data retrieval, manager to manager communications, new MIB definitions, and security enhancements. Other shortcomings are addressed in the OSI version of management protocol. Common Management Information Protocol (CMIP) has a larger set of primitives and a core set of data elements. However, due to its complexity in protocol, CMIP also has disadvantages which help to make the SNMP more attractive. Unfortunately, SNMPv2 and CMIP have not enjoyed wide acceptance in an industry. Note that this thesis will only concern itself with Version 1 (SNMPv1).

SNMP is well suited for hardware interfacing because of how it was designed. Abstract Syntax Notation 1 (ANS.1) and Basic Encoding Rules (BER) were used when developing this protocol. Abstract Syntax Notation is a machine independent high level data definition language; however, it does not state how this data is to be stored or encoded. A subset of ASN.1 is used to describe SNMP messages as well as all fields in the MIBs. Basic Encoding Rules (BER) are used to define how the data types of ASN.1 are to be encoded and later transmitted. It is this machine independent data type definitions and their corresponding encoding that makes this protocol so well suited for interfacing. If all parties agree to predetermined data types and encoding schemes, then communications between dissimilar micro-processor based equipment is greatly simplified. Problems such as big-endian / little-endian, dissimilar variable lengths and types, and improper hardware handshaking are greatly reduced.

The reader should now have enough information on this protocol to understand the theory behind how SNMP is being used in this thesis. As stated previously, SNMP will be employed for hardware control instead of network monitoring and management. The following figure briefly describes the concept behind the use of SNMP in this thesis.

SNMP for Hardware Interfacing

- SNMP Agent program is embedded in the controlled equipment.
- SNMP Manager program is embedded in the controlling equipment.
- Manufacturer supplies MIB for each of its products that supports this protocol.
- Controlling equipment reads MIB for control and monitor objects.
- SNMP Manager sets or resets MIB objects (SET REQUEST) causing the SNMP Agent to control broadcast equipment.
- SNMP Manager reads MIB objects (GET REQUEST) for machine status (such as tape timecode).
- SNMP Agent is doing all the work (similar to distributed processing).
- SNMP Manager has no knowledge of how requested operations are being handled by the Agent program.

Figure 9: How It All Works

Chapter 3

PROCEDURE

A stable operating platform was needed to begin this project, and this became the first objective. I originally started this project working with a UNIX based system. I wanted to use an existing and fully developed SNMP toolkit, as it was not my intention to develop the needed software drivers for this protocol. I was not trying to prove that I could write the code for SNMP; I needed only to use this protocol as a message carrier. The actual interface software for the various hardware devices would be done within the agent programs. The manager programs would simply provide a user interface; however, in actual practice of my concept, this function would be built into the controlling equipment. I was also concerned about using a non-standard version of this protocol. Had it been proven that the implemented version of SNMP was somewhat flawed, it would have negated my final results. Therefore, I attempted to locate a fully developed and tested API of this protocol. Unfortunately at the time, I was unable to find a commercial UNIX based product that fulfilled my needs; however, I was able to find a TCP/IP and SNMP tool-kit (from Dart Communications) for the windows operating system. This dictated my development to be done on this platform.

Having chosen an SNMP API, I then needed to choose a software compiler. I could have purchased the Microsoft Visual Basic version of this tool kit from Dart Communications; however, I felt that the language C would provide more flexibility. With this in mind, I purchased the C++ version of this product and used Microsoft's Visual C++ as my main compiler. This one decision cost me approximately one extra semester in time. I was forced to teach myself Window programming using the Microsoft's Software Development Kit (SDK) due to restrictions within the SNMP toolkit from Dart Communications (see appendix).

I also utilized two other products while developing this thesis. One was the "Windows Standard Communications Library" by MarshallSoft Computing, Inc.. This is a communications library for use with windows compilers, and it's one of the best that I have ever used. It should be noted that this company has a version for DOS as well. The second product was a PicNet Networkable Module from Software Interphase. This is a hardware device which converts serial input to parallel output. It was used to help interface between the PC based proxy agent and the audio/video vertical interval switcher. Both of these products are described in some detail in chapter 6 and the appendixes. Other less notable tools used in this project are described as well.

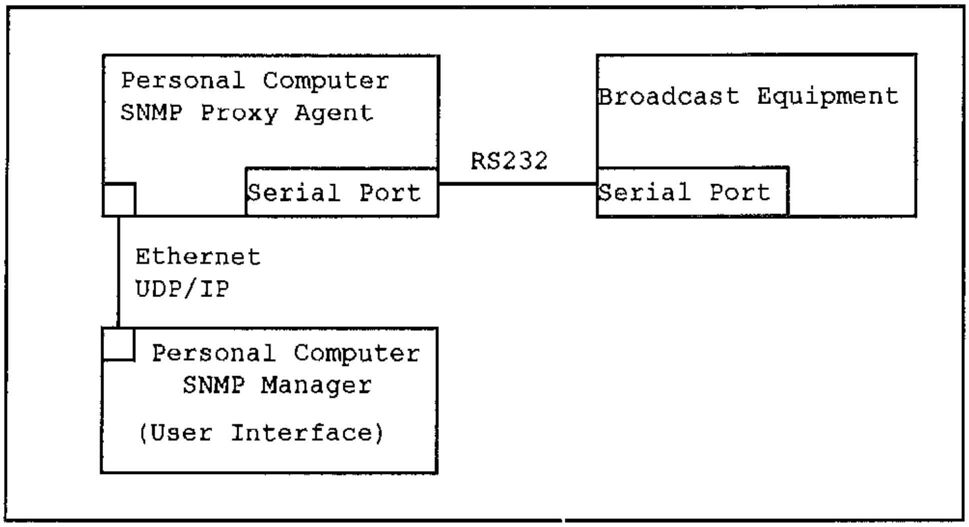


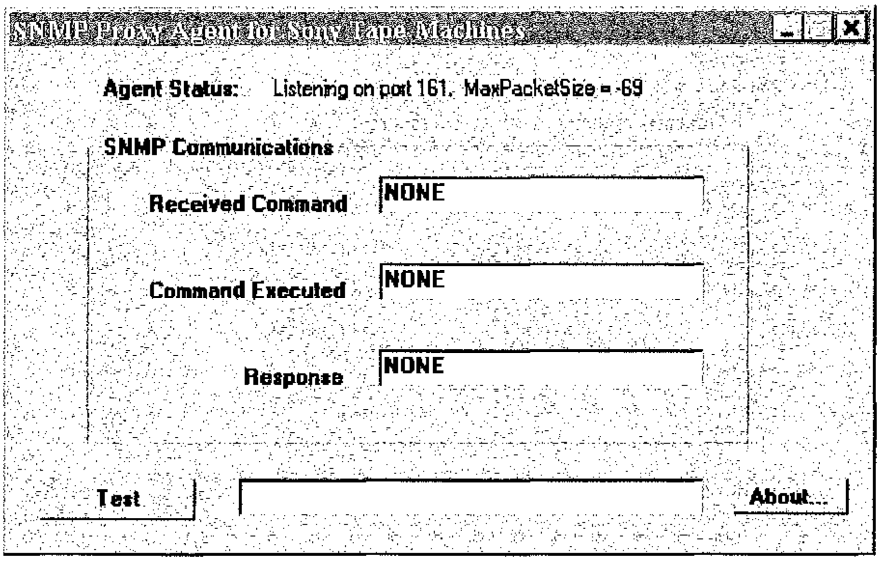
Figure 10: Equipment Interconnection for Testing

Two personal computers (PCs) were used to develop the software. One computer would act as a proxy agent for the equipment to be interfaced, and the other would act as the manager with a graphical user interface. Both units were networked together using Boca 10BaseT Ethernet cards. The two computers used the Windows 95 operating system. One PC was a 133 MHz Pentium based machine and the other was a 120 MHz 586 based unit.

As previously stated, I used a PC as a proxy agent to interface with the various pieces of broadcast equipment. I could have used parallel I/O, serial I/O or specially designed hardware for the interfacing. Fortunately, many vendors make available some type of serial I/O for computer interfacing with their equipment. Each vendor normally supplies the required codes which will allow another computer to control their product; however, at times this information is not always easy to obtain. I chose to use the serial I/O exclusively; thereby, providing as much software reuse as possible. Remember, the manner in which the proxy agent interfaces to the hardware is not important here. Under normal circumstances, this interface would be an integral part of the product. Again, this thesis only wishes to prove the practical use of SNMP in controlling / interfacing broadcast equipment.

Software development began by inspecting and greatly modifying the provided samples that came with the SNMP API tool-kit from Dart Communications. This provided a good starting point, although the sample programs were used in quite a different manner than the way I wished to use this product. Nevertheless, these sample programs provided a wealth of information and I credit Dart Communications for a number of subroutines used within my software.

Basically the three pairs of programs (manager/agent) for each piece of hardware to be interfaced, are very similar. I used object oriented design methods to take advantage of software reuse between the program modules. The similarity is most prevalent between the three versions of proxy agents. Visually, the three versions look almost identical (see figure 11), but internally they are quite different. It is the proxy agent programs that perform the actual interface between the PCs and the various types of broadcast



Agent Status: Listening on port 161. MaxPacketSize = -69

Received Command NONE

Command Executed NONE

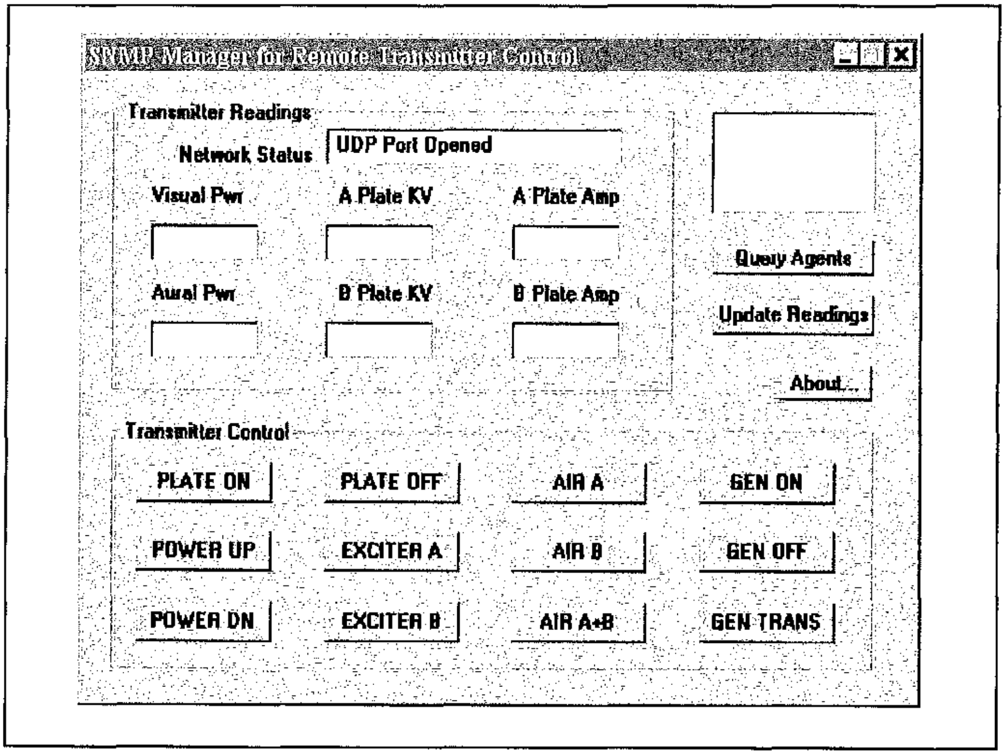
Response NONE

Figure 11: Proxy Agent Screen

equipment. This is where knowledge of the manufacturer's hardware comes into play. The proxy agent programs perform the necessary command translations between the received interface request from the SNMP manager and the appropriate serial I/O command string needed to instruct the equipment to perform the requested operation. Thus, the proxy agent program is doing a great deal of computing, and can be thought of as a form of distributed processing. This distributed processing also helps to keep network traffic to a minimum while providing the necessary hardware interface to the manager program. In other words, the manager program does not need to know anything about how the agent's hardware works. This is the basic idea behind this thesis.

The manager programs are visually, as well as internally, different. These programs rely on user intervention for this demonstration; however, like the agent software, the management operation should be embedded within the controlling equipment. For example, a production switcher would also be running the management software within; thus, negating the need for a proxy manager. Figure 12 shows one of the manager interface screens.

Communications between the manager and agent was accomplished by polling the appropriate agent when necessary. Therefore, the manager program had full control



Network Status: UDP Port Opened

Figure 12: SNMP Management Screen

of the system at all times. The agent program simply performed requested functions.

Not all the functions of SNMP were employed. For example, agent notification to the manager was not implemented, nor was the use of TRAP messages. Acknowledgment feedback was also not used in this project. Visual observation was enough to verify correct operation of the controlled equipment.

The MIBs developed for this project are quite simple. No use of tables were necessary, and only INTEGER and OCTET STRING data types were used. The integer types were used to pass monitoring values from the proxy agent to the manager programs. Octet string data types were used as control switches. Setting one of these would signal the agent program to cause the associated function to be performed on the interfaced equipment. A listing of the MIB objects used in this work is included in the appendix along with examples of other possible MIBs for various broadcast devices. An exhaustive collection of possible equipment interfacing MIBs has not been provided; however, these few selected examples should give the reader a good idea of the different types of equipment that could be controlled through the use of SNMP.

Chapter 4

TOOLS AND DOCUMENTATION

This chapter will concern itself with the various tools used in the development of this thesis. A short synopsis will be given for each tool in this chapter. Additional information on the product will be presented in the appendixes.

The main tool used was Microsoft's Visual C++ compiler version 1.5. The CASE tool was used to create the manager and agent programs. I could have used a later version of Visual C++; however, I wanted to stay in the 16 bit environment for compatibility with Windows 3.xx. I also preferred the overall feel of the editor and debug tool included within the package. The use of version 1.5 caused no problems. See appendix A for further information on this product.

One other compiler was used in this project; however, it was not used to compile or generate source code. I am referring to a MIB compiler and it was only used to check for proper syntax of my experimental MIB modules. I used the SNMP Management Information Compiler - Next Generation (SMICng) from SynOptics. I could have used this compiler to generate source code for implementing the MIBs, but I chose to use the string tables of Microsoft Visual C++ instead. Using the compiler for syntax checking did point out a number of errors in the way I was setting up my MIB as well as identifying various syntax errors. It was a worthwhile exercise. See appendix D for more information.

Along with the two compilers, I also purchased two software toolkits. One toolkit was from Dart Communications and provided the libraries for the SNMP and UDP protocols. As stated previously, it was not my intention to write the SNMP, UDP, or IP protocols. I only wished to use this protocol family in the implementation of this thesis. This toolkit provided the necessary C++ classes for use with the SNMP protocol. The other toolkit was an asynchronous communications library for use with windows. This product came in the form of a Dynamic Link Library (DLL), and was compatible with the Windows API provided by Microsoft. This library provided the necessary APIs for the serial interface between the agent computer and the broadcast equipment. A notable feature of this library is the high baud rate which can be obtained. It supports baud rates as high as 57600 with any word size. I needed this feature because the Sony tape machine required a baud rate of 38800. More information of both of these products can be obtained in the appendixes. See appendix B for Dart Communications and appendix C for MarshallSoft.

As stated previously, vendors usually make available the information on any software protocols needed to interface a computer with their equipment. Sometimes this is provided with the equipment documentation; unfortunately, this is often not the case. The information must be obtained separately and often purchased. The Sony protocol is an example of this. First I had to locate the information and then it had to be purchased. The first obstacle proved to be much harder than the second. It seems that a request for this type of information is not common, and I had a very hard time obtaining the part number for the product. To save the reader from having to experience the same ordeal, I have included all necessary information needed to obtain a copy of this manual in appendix G. This manual describes the Sony Remote-1 protocol for the BVW-10, BVW-11, BVW-15, BVW-35, BVW-40, BVW-60, BVW-65, BVW-70, BVW-75, and BVW-96 models of tape machines. Through experimentation, I found the protocol to work fairly well with other Sony models as well. For example, many commands worked well with the BVU-800 videocassette recorder. Of course, other books and documents on the subject of Simple Network Management Protocol were obtained as well; however, these are given in the reference section of this document. This manual is described here and in the appendix due to the necessity of the contained information and due to the problems obtaining the document.

Two additional hardware devices were used to help with the hardware interfacing between the proxy agent computer and the targeted broadcast equipment. One device used was a RS232-to-RS422 topology converter. It was used to convert the RS232 serial output from a computer to the required RS422 input of the Sony tape machine. The product was designed for Sony tape machines and works well with the required high baud rate. The second device was a type of serial-to-parallel converter. The PicNet module provides eight separate relays for output control. Its input is a serial command stream which directs the addressed module to set or reset targeted relays. This device was used as an interface between a computer's serial port and the audio/video switcher. It was used in a similar fashion to a General Purpose Interface (GPI). Like the others, more information on both these devices can be located in the appendixes.

The only other equipment needed for this project is a couple of network interface cards and the appropriate cabling. I chose to use a couple of Boca 10BaseT Ethernet cards which were NE2000 compatible. Instead of wiring hubs, I simply used a special "twist cable." This cable is easy to make and is nothing more than a cable with the pairs swapped at each end.

Chapter 5

DUPLICATION OF EXPERIMENT

This chapter will focus on instructing the reader in recreating the experiments done in this thesis. This chapter should provide a good starting point for anyone wishing to continue with these studies.

The software source and executable code is provided on the enclosed floppy disk for the reader's use. For verification of the experiments, only the executable files should be necessary; however, for continuation of the work, the C/C++ source code will be needed along with copies of the Specialty Tookkit from Dart Communications and the MarshallSoft Windows Standard Communications Library. This should not prove to be a problem; trial copies of both toolkits are available from the vendors (see appendix). I grant permission for use of my source code to anyone wishing to further this work or to simply experiment with SNMP.

5.1 Verification of Experiments

Recreation of the original experiments requires the executable files along with the two runtime files named Pl6help.exe and Wsc.dll. As stated previously, all files are provided on the accompanying disk with this thesis. Two personal computers running Windows 95 or 3.xx will also be necessary. The original experiment communicated using TCP/IP on Ethernet; however, Ethernet is not a requirement for this project; just a WinSock version of TCP/IP. The experiment should work even if SLIP or PPP is used between machines; however, this has not been verified. Note that both computers will need to have a fixed IP address. RS232 topology was used for communications between the proxy agent computer and the controlled broadcast equipment (see accompanying figure). The actual protocol used between the proxy agent and interfaced equipment is device related.

The main problem for the reader in duplicating this experiment will be gaining access to the hardware which was interfaced. Oddly, obtaining access to a Sony tape machine

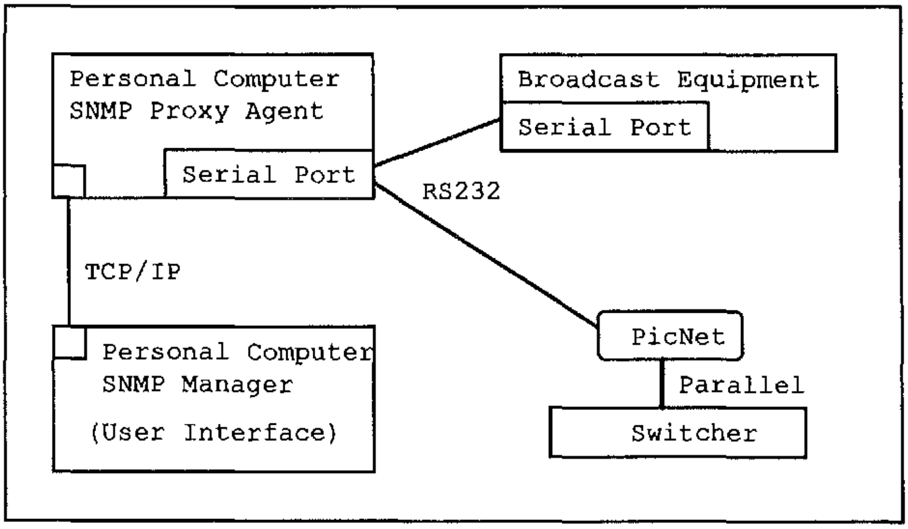


Figure 13: Equipment Setup

may prove to be the easiest. These tape machines are very popular in the academic environment as well as the broadcasting and production industry. Although a Sony BVU-800 was used with this project, any of the Beta series machines should work due to the similarities in the Sony remote protocol between the various models. Of course, this is not guaranteed. The PicNet can be easily obtained from the vender (see appendix) and then used to interface any other device which uses a simple General Purpose Interface (GPI). Unfortunately, the transmitter remote control will probably not be accessible to the reader. The unit was designed and built by this student in partial fulfillment of the requirements for an electrical engineering degree. In other words, only one of these devices exist, and Post NewsWeek (WJXT) has it. However, the documentation for the project should still be on file in the Department of Engineering of the University of North Florida. This document contains all necessary information on the hardware and software needed to understand the internal operation of the unit, or even duplicate the project.

After making all necessary hardware connections, the appropriate software may be executed on the computers. The software programs work in manager/agent pairs. One pair is for the Sony tape machine interface (Sony a/m.exe), one for the transmitter interface(xmit a/m.exe), and one for the switcher interface(switch a/m.exe). Simply choose the appropriate pair depending on what hardware the system will be interfacing. The agent program should be installed on the computer which physically connects to the interfaced device. The manager program is installed on the other computer. Program operation is described in its own chapter.

5.2 Modification of Experiments

As indicated above, the easiest way to experiment with controlling other types of equipment is with the PicNet Networkable Module (see appendix). The necessary proxy agent source code for this module is already written and compiled. All that remains is to connect the eight relays to an appropriate GPI on the equipment needed to be controlled.

For further studies with this project, the software source code must be modified. Due to the properties of Object Oriented Design, this should not be a great burden. Using the Visual C++ compiler will also greatly simplify window reconstruction. All source, definition, and project files are included with this thesis. Note that Visual C++ version 1.5 was used. If a higher version is to be used, the project and definition files may need to be recreated. It is also worth noting that the Microsoft's Foundation Class library (MFCs) was not used. Instead, functions and classes from Microsoft's standard 16 bit API were employed.

5.3 Proxy Agent Modification

The agent program can be easily modified for other types of I/O interface (serial, parallel, etc. ...) by simply modifying one C++ class identified as Ccontrol. This class is defined in the file named CLASSES.CPP, and the file titled CCONTROL.CPP has the C/C++ code for all the functions used within the class. Calls to this class can be found in the CAGENT.CPP and SNMPAP.CPP listings. CAGENT.CPP is where the virtual functions of the CSnmpAgent class are described, and an instance of the Ccontrol class is declared and utilized in these member-function calls. SNMPAP.CPP is the source code which is responsible for setting up the main window. An instance of the Ccontrol class is also declared in this part of the program for the "Test" selection.

The use of a serial port for the interface between the computer and the interfaced equipment is not necessary. The parallel port or any other hardware input/output (I/O) may be used for this purpose. Simply modify the Ccontrol class to handle the new hardware, or create an additional class.

If the serial interface is chosen, then modification of this class should be very simple. The member-function call for sending a serial command to a hardware device is Ccontrol::send_cmd(int *cmdptr). This function takes a character pointer to an array of hexadecimal codes which represent the required serial string.

```
int Ccontrol::send_cmd(int *cmdptr)
{
     int i=0, num, count = 0;

     num = *cmdptr;

     for (count=1;count<num+1;count++)
         {
         SioPutc(Port,*(cmdptr + count));
         }

     return 0;
}
```

Figure 14: Routine to Send a Command String

The first hexadecimal code indicates how many bytes to send. The remainder bytes are transmitted to the interfaced equipment, and are machine/function dependent. Thus, by modifying or creating a new string array, control of different equipment or equipment functions is possible. See figures 13 and 14 for examples.

```
        // Sony Tape Control Command Strings

int sony_who[]          =  {3,0x00,0x11,0x11};
int sony_enable[]       =  {3,0x00,0x1d,0x1d};
int sony_play[]         =  {3,0x20,0x01,0x21};
int sony_time[]         =  {4,0x61,0x0c,0x01,0x6e};
int sony_stop[]         =  {3,0x20,0x00,0x20};
int sony_rewind[]       =  {3,0x20,0x20,0x40};
```

Figure 15: Sample Serial Command Strings

The actual MIB object identifiers are located in a string table which can be easily edited by use of the App Studio of Visual C++ (version 1.5). The IsEqual function, located in the file CALLS.CPP, is used to obtain the object identifier from the string table and compare it to the received identifier.

The Ccontrol class is the main area of interest for modification of the agent program. The remainder of the code will require little modification.

5.4 Manager Modification

The manager program will be a little more difficult to modify. Most of the needed changes will be with the main window layout. Window control objects (buttons) are associated with a member-function calls of the class SnmpManager. The function MainWnd_OnCommand will contain the code where the various buttons are decoded and used to send the correct SNMP message to the agent program. An example is given below.

```
     case IDC_PLAY:

           fObjectType = SNMP_OCTET_STRING;

           _fstrcpy(szObjectId, GetString(hinst,
                   IDS_SonyPlay));

           lpszObjectId=szObjectId;

           lpszObjectValue = "1";

           pav->pSnmpManager->state =
                   SNMPM_STATE_SETOBJECT;

           pav->pSnmpManager->SendSetRequest(

           pav->pSnmpManager->szRemoteHost, IPPORT_SNMP,
                   "public",1,1, (LPSTR FAR*)
                   &lpszObjectId, (LONG FAR*)
                   &lpszObjectValue, &fObjectType);

          break;
```

Figure 16: Example SNMP Manager Message Routine

Like the agent program, the MIB object identifiers are located in a string table. The function GetString, in file CALLS.CPP, is used to obtain the MIB object identifier from the string table for use with the member-function call to the SnmpManger.

Note that the SNMP object type being sent must match the type declared within the agent program. In the example above, note that the object type is an octet string.

Changes in these two areas should be all that is really necessary to use the manager program with a modified agent program. The rest of the code need not change much, if at all.

Chapter 6

PROGRAM OPERATING INSTRUCTIONS

This chapter deals with the operation of both types of programs. Note that basically all three program pairs (manager/agent) work the same.

As stated previously, all files are provided on the accompanying disk supplied with this thesis. Two personal computers running Windows 95 or 3.xx will also be necessary. The two computers communicate using WinSock TCP/IP on Ethernet. Ethernet is not a requirement for this project; just TCP/IP. Note that both computers will be required to have a fixed IP address.

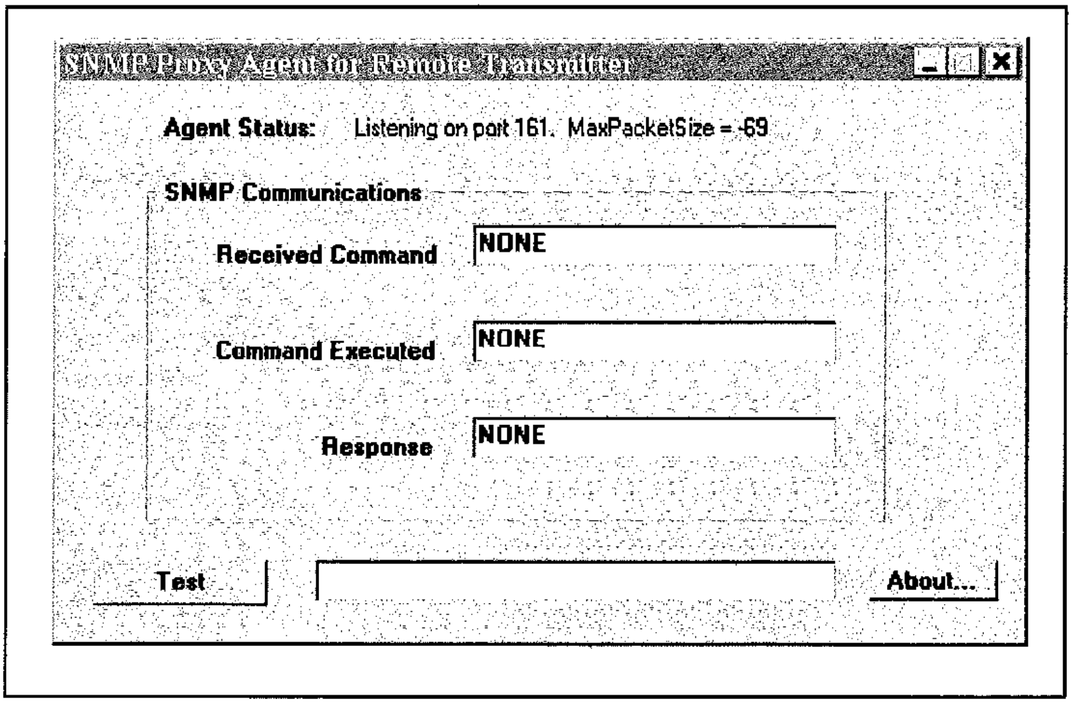
RS232 topology is used for communications between the proxy agent computer and the controlled broadcast. Any cables necessary will have to be supplied by the user.

Program termination presents no problems and does not require any special shutdown sequence. Terminate the programs in any fashion when operation is complete.

6.1 Agent Programs

The operation of the agent program is quite simple and somewhat limited. User interaction with the agent program is not really necessary, but the program can be used to monitor operations and SNMP messages between the two computers. Status windows on the program screen are provided for this purpose.

The agent programs also have a test feature which exercises the interfaced equipment in some fashion. This proves that the proxy agent computer is talking with the device. Note the test button in the figure below. All three agent programs operate in a similar fashion.



Agent Status: Listening on port 161. MaxPacketSize = -69

Received Command NONE

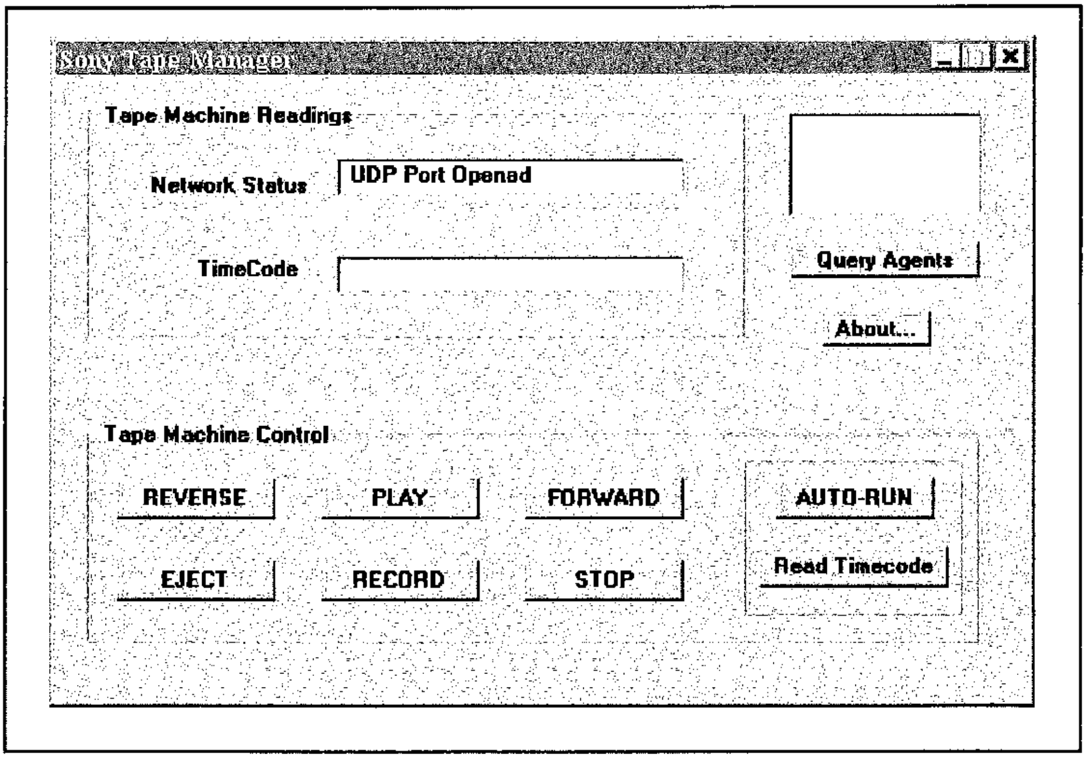
Command Executed NONE

Response NONE

Figure 17: Agent Screen

6.2 Manager Programs

Operation of the manager programs is moderately self explanatory. The only exception to this is how one makes the initial contact with an agent program. The process is simple. Note the button labeled "Query Agents" on the window screen. This is used to query any agents on the network. If any agent programs are listening, they will respond with a message indicating their IP address. The address will appear in the edit window above the button. "Clicking" on the IP address of the machine with the desired proxy agent program will complete the connection.



Network Status: UDP Port Opened

Figure 18: Manager Screen

Once connected with the agent program, operation of the manager program is accomplished by merely doing a "point and click" on the desired function indicated by the control buttons displayed in the window.

Chapter 7

RESULTS AND CONCLUSION

All three systems performed as predicted, and I was quite pleased with the overall response of the systems. The transmission speed of the data packets over 10 MBit Ethernet did not prove to be a significant factor in system reaction time.

I was most concerned with reading timecode (time markings recorded on the video tape used for location purposes) from the tape machine. Had the transmission time been significant, the reported timecode would have been inaccurate; however, this was not the case. Also note that the program only polled the tape machine for a timecode reading when the user initiated the event. Perhaps constant timecode updates would prove to be a problem; however, due to the distributed processing nature of the agent program, this situation could easily be avoided. For example, instead of the manager program searching for a certain timecode by constantly requesting a time reading, the agent program could be sent the desired time and it could perform the search.

The transmitter control system and the video/audio switcher responded well also, but these devices were not real-time critical.

The system was developed and mainly tested using two personal computers networked together via 10BaseT Ethernet. The system was also tested on an existing LAN (10BaseT) which has about 80 client machines attached. The increase in network traffic did not appear to cause any degrade in the performance of the tested system.

As this thesis has demonstrated, SNMP can be used as an interfacing protocol. Although it may not be desirable for critical real-time applications, it definitely can be used

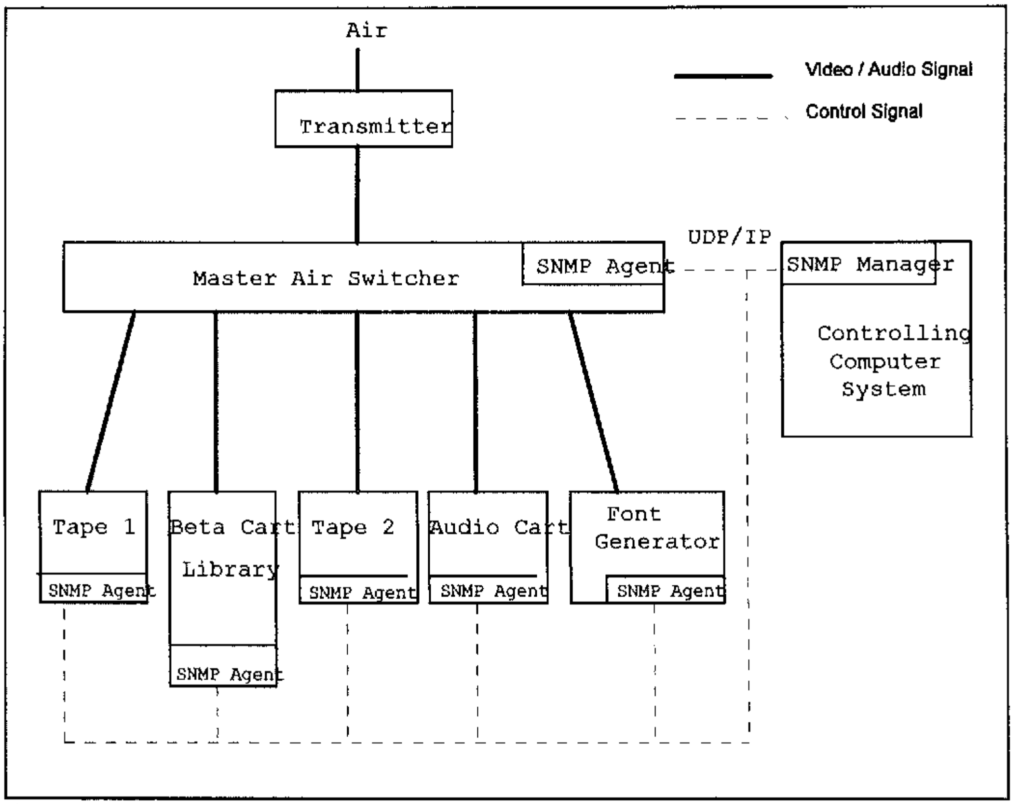


Figure 19: Control of Master Air Switcher and Sources

with the majority of machine control purposes in the typical broadcast environment. This system could be employed in controlling devices such as master-control switchers (Fig. 12), production switchers, audio consoles, audio cart machines, house routers, character generators, transmitters, cameras, robotics camera pedestals, tape machines, satellite receivers and antennas (dishes), and various types of test equipment. In addition, this protocol could be used to interface the specialized text editing computers within a news department with various types of production equipment. This would enable these computers to be used for controlling scheduled tape recordings, signal routing, creation of graphic playback list, character generator loading with story slugs, and the like. Response time needed for many of these types of operations is well within the capabilities of this protocol.

The use of the PicNet serial-to-parallel converter points to another use of this interfacing scheme. General Purpose Interfaces (GPIs) are often used to enable simple contact closure control of equipment. For example, an editing machine may provide several GPIs which can be set or reset on editing events such as a video cut or at a certain timecode. This contact closure could trigger a font generator or, perhaps, an audio cart. Although simple, GPIs are often used and there never seems to be enough GPI connections. Perhaps SNMP could be used as a GPI expander

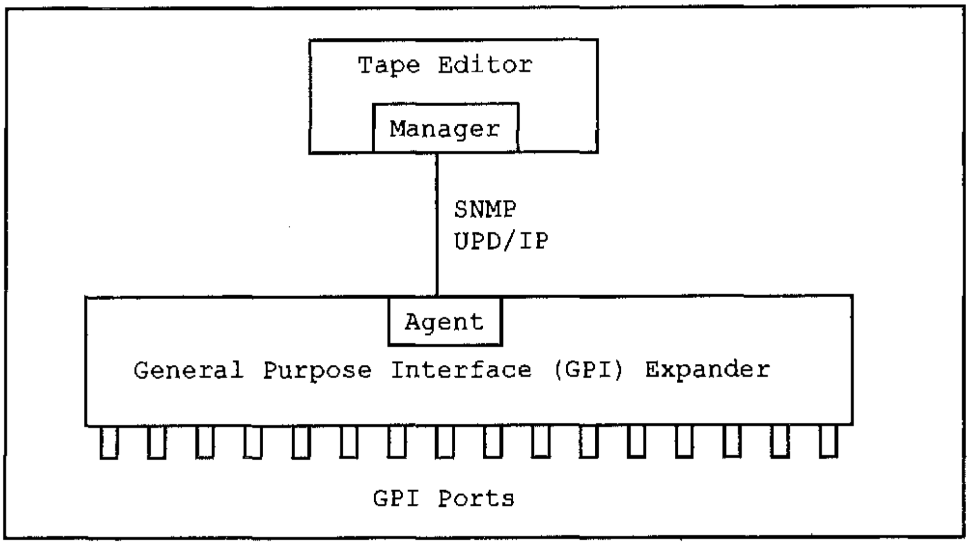


Figure 20: GPI Expansion Example

as well. For example, instead of requiring the equipment to have a physical connection for each GPI, it could simply support many virtual GPIs via this protocol. A GPI expander unit housing the agent program would supply the necessary I/O ports. The originating equipment (tape editor in figure 13) would simply modify the required port variable of the corresponding GPI MIB. The expander would set or reset the required port. This is basically how the PicNet module is being used in this project.

This thesis has shown that the SNMP protocol can be used for more than just monitoring network equipment; it can be used for complex control purposes as well. Because of Abstract Syntax Notation 1 (ASN.1), Basic Encoding Rules (BER), and the Management Information Base (MIB), dissimilar micro-processor based hardware can easily communicate. The use of MIBs would allow for easy software configuration for each device being controlled. Combined, these features make SNMP a possible solution for the broadcast industry.

A better solution might be to design an interface protocol explicitly for the industry. I recommend that the ASN.1, BER, and MIBs be employed if this is done. Perhaps this thesis has demonstrated the framework of a future protocol.

REFERENCES

[Comer94]
Comer, Douglas E. and Stevens, David L. Internetworking with TCP/IP. Prentice Hall, 1994.

[Feit95]
Feit, Sidnie. SNMP, A Guide to Network Management. McGraw-Hill, 1995.

[3M]
3M Company, Mincom Division. Video Bridging Switcher Instruction Manual. Camarillo, CA.

[Parker94]
Parker, Timothy. Teach Yourself TCP/IP in 14 Days. Sams, 1994.

[Perkins97]
Perkins, David and McGinnis, Evan. Understanding SNMP MIBs. Prentice Hall, 1997.

[Peterson95]
Peterson, David M. TCP/IP Networking, A Guide to the IBM Environment. McGraw-Hill, 1995.

[Piscitello94]
Piscitello David M. and Chapin, Lyman A. Open Systems Networking TCP/IP and OSI. Addison-Wesley, 1994.

[Satz96]
Satz, Greg L. "Origins of SNMP." The Simple Times (March/April 1992), pp 2-5. Online. Internet. 3/16/96.

[Sony88]
Sony Corporation. Sony Protocol of Remote-1 (9 Pin) Connector. Japan, 1988.

[Stevens94]
Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Addison_Welley, 1994.

RFCs

RFC 1089	SNMP Over Ethernet.
RFC 1155	Structure and Identification of Management Information for TCP/IP Based Internets.
RFC 1157	A Simple Network Management Protocol.
RFC 1158	Management Information Base Network Management of TCP/IP Based Internets: MIB-II.
RFC 1212	Concise MIB Definitions.
RFC 1213	Management Information Base for Network Management of TCP/IP Based Internets: MIB-II.
RFC 1215	A Convention for Defining Traps for use with the SNMP.

APPENDIX A

Microsoft Visual C++ Version 1.5

While being one of the most complicated tools I have ever used, it is also one of the best. This product is a complete Windows development CASE tool employing Object Oriented Programming (OOP) in the C++ language. The product includes a compiler, project manager, build utility, browser, debugger, resource editor, a text editor, a graphics editor, and a couple of wizards.

The AppWizard is a program used to help create a skeleton Window source program. The ClassWizard is used to create new classes, and to browse existing ones. The Wizards greatly help in the development of a Microsoft Foundation Class (MFC) OOP program, but it does nothing for a program developed using only the Software Development Kit (SDK) functions.

The AppStudio is used to create graphic interfaces, and to assign variable names to the various data items on a screen. This tool is also used to generate String Tables, and these are tables used to create string constants. I used these tables to keep the MIB object identifiers for the manager and agent programs. The AppStudio can be used with MFC and SDK programs equally.

As indicated above, this compiler can be used with Microsoft Foundation Classes as well as the Software Development Kit classes; however, it can also be used on ANSI and DOS code as well. One of the project options is to create a QuickWin program. This is a program which will take DOS or even UNIX ANSI C/C++ code and creates a program which can be executed in the Windows environment. QuickWin is great for porting old DOS or UNIX programs to Windows; very little code modification is necessary.

The debug capabilities of this tool is worth the price of the entire package. The ease with which one can set break points, monitor variables and arrays, and step through code can save hours of debugging time.

This product's massiveness and flexibility are also one of its disadvantages. The environment is complex and somewhat difficult to master for someone who has not been introduced to the OOP and Windows environment. Remember that event-driven coding is used with this product unlike the normal sequential coding practice. Skeleton Window programs can be easily generated with little effort; however, one quickly runs into problems when options are added to the resultant code. The concepts and language syntax are not where the problems lie. In my opinion, the difficulty is with the necessary documentation. Even if one has mastered C++, this product cannot be easily used without hours of study from more that one source. It should be noted that Microsoft has made a good effort to remedy this situation. Many examples and a good deal of documentation is provided with the package. An understanding of the Object Oriented Programming methodology will also greatly help to understand thought process behind the great CASE tool.

If one is familiar with the Borland family of compilers, then this tool may seem to be unnecessarily complex; nonetheless, the extra effort will be rewarded. This is one of the best compilers that I have used.

APPENDIX B

Dart Communications Specialty Toolkit (PowerTCP)

Dart Communication produces a number of TCP/IP based toolkits. They support the 16 and 32 bit versions of the Microsoft C++ compiler, Visual Basic, Delphi, PowerBuilder, and ActiveX. The Standard Toolkit provides libraries for TCP, TELNET, FTP, SMTP, POP3, and VT220 emulation. The Specialty Toolkit used in this thesis has libraries for UDP, TFTP, and SNMP. This is the toolkit used to provide the necessary SNMP protocol with UDP support in this thesis.

PowerTCP provides five type of interfaces. For the C compiler, C++ class libraries and DDL are provided, and for the other languages mentioned above, VCL components, Visual Basic custom controls and OLE custom controls are provided. The Specialty Toolkit provided two classes which were used in this thesis. The classes and their functions are presented in figure 21. All components interface with a Winsock layer. Although 16 and 32 bit C++ versions are available, I used the 16 bit for compatibility between Windows 3.xx and Windows 95.

The package comes with a number of very good sample programs written for use with Visual C++; however, the code was written using the Software Development Kit (SDK) instead of the Microsoft Foundation Classes (MFC). I bring this up

CPowerUdp:

Public Member Functions:

CPowerUdp ~CpowerUdp Connect Close Send State

Virtal Member Functions:

ExceptionEvent ConnectEvent RecvEvent SendEvent

Constants

PT_EXCEPTION PT_STATE

CPowerSnmp:

Public Member Functions:

CPowerSnmp ~CPowerSnmp Connect Close State Trap
SendGetNextRequest Send Get Request SendSetRequest
SendGetRequest SendTrap

Virtal Member Functions:

ExceptionEvent RecvSnmpEvent RecvTrapEvent SendEvent
ConnectSnmpEvent RecvTrapEvent

Constants

PT_EXCEPTION PT_STATE SNMP_ERROR SNMP_TYPE
SNMP_OBJECT_TYPE SNMP_TRAP

Figure 21: SNMP Classes and Functions

because I had difficulty getting the SNMP Classes to work with a window program based on MFCs. I called technical support and received very little help. I was finally told that the product did not support the MFC environment. I really don't believe this, and I'm sure that if I were more experienced with the MFC environment, I could have made it work. Therefore, I created the necessary windows interface in my projects using SDKs. This did cause some additional development time due to my unfamiliarity to these classes and procedures.

I have no problem recommending this product. The preceding problem was the only one found with this product; it performed flawlessly. Documentation was adequate and supplemented the provided example code very well. The classes were very straight forward and easy to use. Purchase of a toolkit grants the programmer to royalty-free use of the product. The following four pages contain the licensing information from the Dart Communication toolkit. The text was taken from their provided documentation without modification. The last section (PowerTCP Sample Application Documentation) references use of the example programs.

Dart Communications can be reached at the location provided below. A thirty day trial package can be downloaded for many of their products from the company's Web site. Products not listed on the Web site may still be offered as a trial package if you write or call the company.

Dart Communications
6647 Old Thompson Road
Dewitt, NY
Tel: (315) 431-1024
Faxl: (315) 431-1025
E-mail support@dart.com
http://www.dart.com

PowerTCP Licensing Options

May 23, 1996

Version 2.0

Thank you for purchasing this PowerTCP Toolkit. It includes one Development License and one Royalty-Free License for your development and distribution purposes. Additional licensing is no longer required, but you may find it desirable for completely transparent operation or if software maintenance is desired.

Dart Communications offers 3 licensing options to meet the needs of most developers. Just choose the one that's best for you!

PowerTCP Royalty-Free License (included with every toolkit)

All applications written with PowerTCP Toolkits version 2.0 will run in any WinSock-compliant environment with no special set-up or licensing required. A minimized icon is visible when your application is using PowerTCP that identifies your company as a PowerTCP Licensee.

This license is adequate when PowerTCP product identification does not conflict with application goals and software maintenance is not required.

PowerTCP End-User License (1 is included)

A serial number is installed on each runtime system in the local WIN.INI file (the presence of this serial number disables PowerTCP product identification so that PowerTCP operation is completely transparent). This license applies to all protocols and development interfaces implemented in the PowerTCP product line.

This license is appropriate for limited numbers of deployed applications where PowerTCP product identification is not desired.

PowerTCP OEM Partner Subscription (available separately)

For ISVs and other customers with stringent support requirements, the OEM Partner Subscription establishes Dart Communications as your network programming support arm. For a fixed annual fee, this license provides you with:

+ the right to use any of the libraries for the protocol being licensed (e.g. right to update from VBX to OCX with the only cost being a new OCX Toolkit)
+ priority software maintenance and support with resolution within 5 working days of notification
+ free Toolkit of choice upon annual renewal
+ optional escrow account for software source
+ time & material guarantees for engineering support
+ access to Dart's expert technical support team

Thank you for your interest in our line of PowerTCP products. Please contact me if I can be of assistance in any way.

Sincerely,

Allison G. Smith
Account Manager

TECHNICAL ADDENDUM

Installation

When the PowerTCP Toolkit is installed on your system, P16HELP.EXE is stamped with your company name when SETSN16.EXE is run during the installation process (you can run SETSN16.EXE at any time to modify this stamp...just include the full path and file spec for SETSN16.EXE as a command line parameter). P16HELP.EXE (the License Manager Application) can be run interactively to manage the license that is currently installed on your system. It will also identify your company as licensee of PowerTCP. SETSN16.EXE can be run without a command line parameter to modify or install the End-User or Trial License in your WIN.INI file.

For all licenses except OEM Partner Subscriptions, P16HELP.EXE is loaded when you use a PowerTCP component. For this reason, P16HELP.EXE must be distributed as runtime support along with the PowerTCP component(s) you are using. PowerTCP will not operate unless this file is included with your distribution. If you do not wish to distribute P16HELP.EXE with your product, then you should investigate our OEM license.

PowerTCP Royalty-Free License

When P16HELP.EXE is loaded by your PowerTCP component and no End-User License is found, then the Royalty-Free license is assumed. A minimized icon is then shown that, when restored to normal window size, identifies the PowerTCP product and your company name as licensee.

PowerTCP Trial License

The PowerTCP 30-day trial license, available by contacting Dart's sales office, is inserted as a WIN.INI entry as follows:

```
[Dart Communications]
PowerTCP License=30-xxxxxxx-xxxxx
```

PowerTCP will not operate after the 30-day period expires. You should contact Dart for a PowerTCP Toolkit.

PowerTCP End-User License

The PowerTCP End-User license, included with all PowerTCP Toolkits and available separately, is inserted as a WIN.INI entry as follows:

```
[Dart Communications]
PowerTCP License=100-xxxxxxx-xxxxx
```

PowerTCP will operate transparently on your behalf, with no PowerTCP icon visible.

PowerTCP OEM Partner Subscription

Dart will issue you a serial number that disables all license checking. P16HELP.EXE is not needed as run-time support, and need not be distributed with your product. By using the OEM Partner License, PowerTCP operates transparently on your behalf, with minimum resource utilization.

License Summary

By offering these license options, Dart Communications hopes to provide a good licensing solution for most situations. Thank you for using PowerTCP, the most advanced TCP/IP Toolkit available today!

Copy of Shrink-wrap License Agreement

POWERTCP SOFTWARE LICENSE AGREEMENT

This is a legal agreement between you (either an individual or entity) and Dart Communications. By opening the sealed software packet you are agreeing to the terms of this agreement. If you do not agree to the terms of this agreement, promptly return the unopened software packet and the accompanying items (including written materials) to the place you obtained them for a full refund.

GRANT OF LICENSE. This License Agreement permits you to use one copy of the enclosed PowerTCP software program (the "SOFTWARE") on a single computer. The SOFTWARE is in "use" on a computer when it is loaded into temporary memory (i.e. RAM) or installed into permanent memory (e.g. hard disk or other storage device) of that computer. PowerTCP must be licensed for each programmer that uses it. This License Agreement is non-sublicenseable.

COPYRIGHT. The SOFTWARE is owned by Dart Communications and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g. a book or musical recording) except that you may either (a) make one copy of the SOFTWARE solely for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may not copy the written materials accompanying the SOFTWARE.

OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE, but you may transfer the SOFTWARE and accompanying written materials on a permanent basis provided you retain no copies and the recipient agrees to the terms of this agreement. You may not reverse engineer, decompile, or disassemble the SOFTWARE.

TERMINATION. This license is effective until terminated. This license will terminate automatically without notice if you fail to comply with any provision of this license. Upon termination, you must cease all use of the program and return it, including any archival copies of PowerTCP.

LANGUAGE SOFTWARE is defined to be .EXE, .LIB, .DLL, .VBX and .OCX files that you link to your software. You have a royalty-free right to staticly link to or reproduce and distribute LANGUAGE SOFTWARE provided that you: (a) distribute the LANGUAGE SOFTWARE only in conjunction with and as part of your end-user application. As used herein, the term "end-user application" does NOT include development tools, or any application or utility that requires that the user distribute the LANGUAGE SOFTWARE; (b) do not distribute any .LIC license files; (c) do not use Dart Communcation's name, logo, or trademarks to market your software product; and (e) agree to indemnify, hold harmless, and defend Dart Communications from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software product. The LANGUAGE SOFTWARE are those files in the SOFTWARE that are identified in the accompanying materials as required during the execution of your software program.

An "end-user application" is an application that may not be copied or redistributed by the user. An application that will be incorporated into another application is not an end-user application. If all or any portion of your application is redistributed by your customer, your application is not an end-user application. If you want to use PowerTCP in an application that is not an end-user application, you should contact Dart Communications to obtain information on other licensing options.

This Agreement is governed by the laws of the State of New York.

PowerTCP Sample Application Documentation

This document describes the sample applications shipped with PowerTCP. You can learn how PowerTCP works by looking at the samples. You may also use sections of code from the samples in your own software.

All PowerTCP samples are described here. The *PowerTCP Standard Toolkit* includes 10 samples and *PowerTCP Specialty Toolkit* includes 4 samples. All samples are provided with C, C++, Visual Basic, PowerBuilder and Delphi source code, depending on the toolkit purchased.

APPENDIX C

MarshallSoft Windows Standard Communications Library

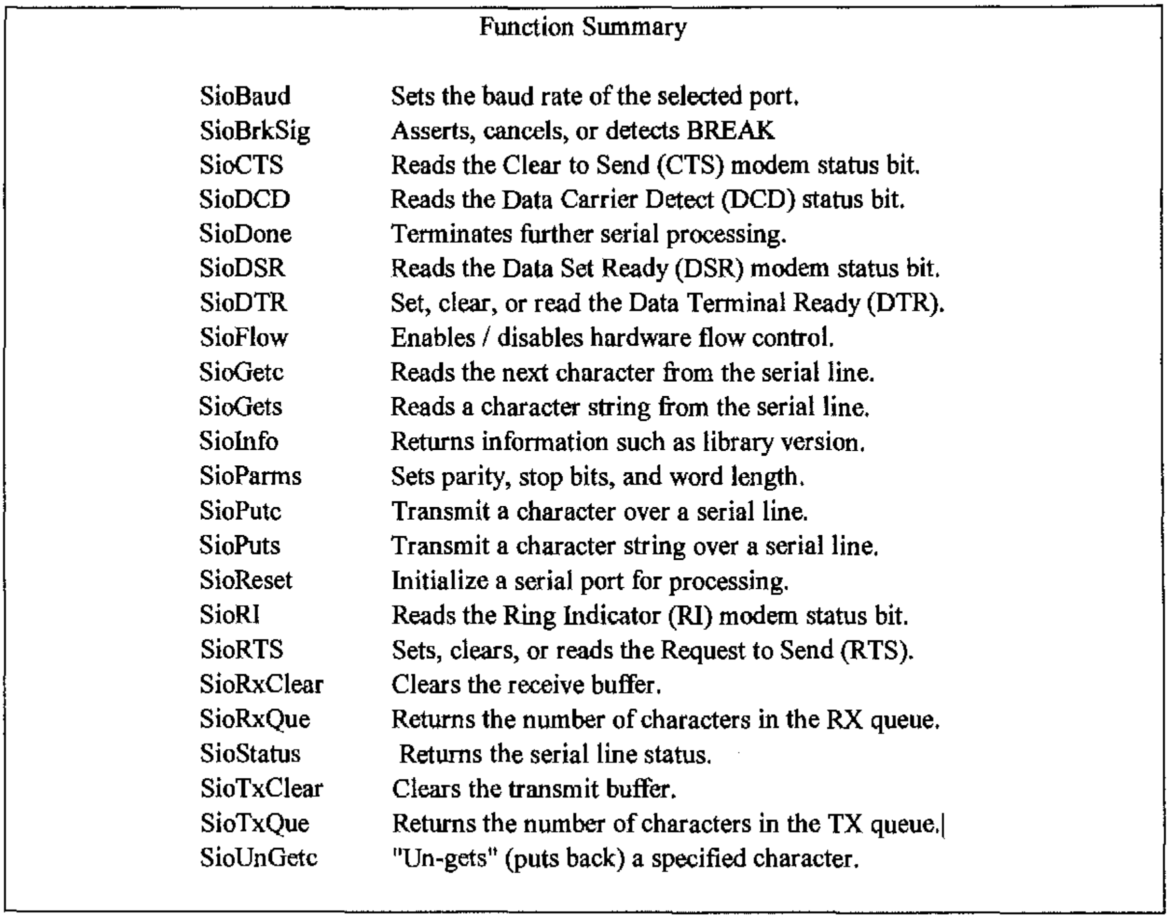
This is an asynchronous communications C/C++ dynamic link library (DLL) for Windows. It uses the standard Windows API; therefore, it is compatible with any other programs which also use the Windows' API. Because it's a DLL, it can support most of the popular C/C++ compilers. Examples, along with the makefiles, are provided for Borland, Watcom, and Microsoft.

The library contains over twenty-five functions for serial communications, and it is very straight-forward to use. Note the following code which can be used to output the message “Hello World” through serial port COM1.

```
SioReset(COM1,128,128);
SioPuts(COM1,"Hello World",11);
SioDone(COM1);
```

Figure 22: Sample Code "Hello World"

Simplicity and flexibility are the main features of this product. The software can handle baud rates of up to 57600, with any word bit combination. The high baud rate capability was the main reason I chose this product. I needed a 38800 baud rate to interface with the Sony tape machine. The Request-To-Send(RTS)line can be set or reset at any time, and the Data-Carrier-Detect(DCD) and Clear-To-Send(CTS) lines can be easily read. Flow control can be either hardware (RTS/CTS), software (Xon/Xoff), or none. Transmit and receive buffer sizes can be set independently, and the buffers can be monitored and cleared at will. All four serial ports can be addressed, and port status can be monitored. The library even came with a special DLL for modem control using the modem AT command set.



Function Summary

SioBaud	Sets the baud rate of the selected port.
SioBrkSig	Asserts, cancels, or detects BREAK
SioCTS	Reads the Clear to Send (CTS) modem status bit.
SioDCD	Reads the Data Carrier Detect (DCD) status bit.
SioDone	Terminates further serial processing.
SioDSR	Reads the Data Set Ready (DSR) modem status bit.
SioDTR	Set, clear, or read the Data Terminal Ready (DTR).
SioFlow	Enables / disables hardware flow control.
SioGetc	Reads the next character from the serial line.
SioGets	Reads a character string from the serial line.
SioInfo	Returns information such as library version.
SioParms	Sets parity, stop bits, and word length.
SioPutc	Transmit a character over a serial line.
SioPuts	Transmit a character string over a serial line.
SioReset	Initialize a serial port for processing.
SioRI	Reads the Ring Indicator (RI) modem status bit.
SioRTS	Sets, clears, or reads the Request to Send (RTS).
SioRxClear	Clears the receive buffer.
SioRxQue	Returns the number of characters in the RX queue.
SioStatus	Returns the serial line status.
SioTxClear	Clears the transmit buffer.
SioTxQue	Returns the number of characters in the TX queue.
SioUnGetc	"Un-gets" (puts back) a specified character.

Figure 23: MarshallSoft Communications APIs

The preceeding function summary (figure 23), obtained from the documentation, is provided to demonstrate the power and flexibility of the product.

The beauty of the toolkit is that it is available for DOS as well, and it uses basically the same function calls. This makes porting code from DOS to Windows quite painless. The product even supports other languages such as Pascal and Visual Basic.

I've used this product for years and recommend it highly. I have had few, if any, problems with the software. The company does supply technical support, but I've never needed it. A trail copy can be downloaded from the company's Web site or BBS. The selling price is more than fair. MarshallSoft can be reached at the following location.

MarshallSoft Computing, Inc.
Post Office Box 4543
Huntsville, Al 35815

BBS: (205) 880-9748
Tel: (205) 881-4630

E-mail: info@marshalsoft.com
www.marshallsoft.com

APPENDIX D

SynOptics (SMICng) MIB Compiler

SNMP Management Information Compiler - Next Generation

The MIB compiler used with this thesis is the updated form of the popular SMIC from Bay Networks. SMIC is still freely available from this company, and the full SMICng version is available from SynOptics. I used a somewhat limited version which came with the textbook Understanding SNMP MIBs. The book is excellent and so is the compiler.

This product can be used on a variety of operating system platforms. SMICng is written in portable C code (source and makefiles available) making it easy to port to various systems. The CDROM has versions for MSDOS/Windows, Sun Solaris 2.x, Sun SunOS4.x, HP HPUX, IBM AIX, and Linux.

SMICng can be used alone to simply check MIB syntax, or it can be used to produce an intermediate output for use with back-end compilers to generate MIB representations in other formats such as schema files, data structures, and code for application development. These optional back-end compilers can be supplied by the vendor or they can be written by the user. The format of the intermediate code is well documented and should prove to be easily parsed. One popular output format from this compiler is the MOSY v7.1.

The MOSY format is part of the ISO Development Environment (ISODE) package.

Operation of the compiler is fairly simple. An include file is created which configures the various options during compile time. The MIB to be compiled is identified in this include file as well. The name of this file is the command line argument to the program call. An example of an include file follows.

```
                     Include File for SMICng Compiler

-- file: videoproc.inc
-- Modules referenced by module VIDEOPROC-MIB

#condInclude "rfc1155.inc" -- RFC1155-SMI
#condInclude "rfc1212.inc" -- RFC-1212

-- MIB module
#pushOpt
-- Remove strict checking
-- Options:
--    C - check size/range present
--    W - don't allow size/range for items in a sequence
--    7 - restrict INTEGER values below 2G-1
--    R - check (in V1) that INDEX objs are read-only
--    S - require (in V2) that IMPORTS be specified for items in compliances
--    B - strong checking for size/range of items in index clause
#removeOpt "C W 7 R S B"

-- Loosen checking
-- Options:
--    4 - allow non-standard access for objects
--    K - allow (in v1) zero valued enums
--    O - allow (in v2) hyphens in labels for enumerated values
--    P - allow (in v2) hyphens in descriptors(identifiers)
--    T - no check (in v2) of proper access for items in groups
--    M - no check (in v2) that all NTs and accessible OTs are in a group
--    F - allow integer/integer32 index items without a range
--    G - allow unused IMPORTS and textual conventions
--    N - no check (in v2) of access of objects in notifications
--    I - use (in v1) the v2 rules for checking ACCESS of index items
--#addOpt "4 K O P T M F G N I"
#addOpt "F" -- allow integer/integer32 index items without a range

#condInclude "videoproc.mib"
#popOpt
```

Figure 24: Sample Include File

All referenced RFC's and other modules must be in a directory pointed to by a system variable or must be in the local directory. The package comes with many stripped and corrected MIBs, obtained from RFCs, for use with source MIBs to be compiled.

The compiler comes with a surprisingly rich set of features. The following is a partial list of features provided by the accompanying text (Perkins and McGinnis, 425).

- Can read Multiple input files.
- Parses MIBs written in the syntax defined by SNMPv1 SMI, concise MIB, and trap format document(RFC1155, RFC1212, RFC1215).
- Parses MIBs written in the syntax defined by SNMPv2 SMI, SNMPv2 Textual Conventions, and SNMPv2 conformance documents (RFC1442, FRC1443, and RFC1444).
- Parses multiple MIB modules in one input stream.
- Check the validity of IMPORTS clauses.
- Resolves textual conventions and checks that their usage is valid.
- Supports use of extended ASN.1 size/range constructs.
- Can create SNMPv1 MIBs from SNMPv2 MIBs.
- Can create MOSY v7.1 .defs and .tcl files.
- Alias assignments for modules and object names.
- Has selective checking of MIB constructs.
- Has extensive MIB syntax checking and can continue syntax checking after most syntax errors.

- Has extensive checking of MIB constructs.
- Has multiple output options.
- Has conditional compiling of MIB modules based on need.
- Can exclude imported MIB modules from outputs.
- Has extensive help via command line options.

Two other utilities are supplied with the package. The first is a MIB stripper to remove or strip RFC documents of the MIB module. Basically, all the supporting text is removed leaving only the module. The second utility takes the intermediate output from the compiler and uses it to create an HTML document for viewing with a Web browser. This utility was used to generate the HTML representation of MIBs created with this thesis (see accompanying disk).

The compiler can be obtained from the following Web site or by purchasing the text from the publisher.

TEXT:

Understanding SNMP MIBs
Prentice Hall PTR
Upper Saddle River
New Jersey 07458
www.prenhall.com

ISBN 0-13-437708-7

WEB SITE:
www.snmpinfo.com

APPENDIX E

Imagine Serial Converter

This is a hardware product used to convert between RS422/485 and RS232 topology. The device is tailored for Sony tape machines, and is used to allow a standard RS232 serial communication port to be used to interface to the RS422 serial port of the tape machine. The unit has a 9 pin adapter (DB9) for use on the tape machine, and a 25 pin adapter (DB25) for the RS232 side. The electronics are contained within the DB25 connector and power is obtained from the RS232 line (less that 2ma). An optional power transformer is provided for situations requiring additional power. The unit supports Sony protocol which runs at 38800 Baud.

The Serial Converter can be purchased from the vendor directly.

Imagine Products Inc.
581 South Rangeline Road, Suite B-3
Carmel, Indiana 46032

Tel: (317) 843-0706

APPENDIX F

PicNet Networkable Modules

This product was designed for interfacing electrical power equipment to any computer which has a RS232 serial port. Each model uses eight relays for controlling current loads, up to ten amps, for eight different devices, and multiple modules (255 Max) may be concatenated together to allow controlling a greater number of devices.

Serial RS232 topology is used for communications between each module and the controlling computer. The units are configured to 9600 Baud, 8 data bits, 1 stop bit, and no parity. Also note that Data Terminal Ready (DTR) must be low.

The protocol is simple; READ, WRITE, and POLL are the only three types of commands used with this device. The WRITE command is used to set or reset the various relays within an addressed module. The READ command is used to determine a module's status, and the POLL command is used to determine the module's model number and revision level of its design. Each transmitted instruction is composed of a four byte packet. The first byte determines the type of command to be issued (READ, WRITE, or POLL). The second byte represents the destination station's address, and the third byte is simply a zero. The fourth byte is determined by the type of command to be sent. For a WRITE command, the fourth byte will represent a control mask which is used to determine which relays will be set or reset within the addressed module. For READ and POLL messages, the fourth byte is set to zero. For each byte sent, the module will return an acknowledgment byte.

This product was used to interface the audio/video switcher used in this thesis, but it could have been easily used to interface any device which uses a remote General Purpose Interface (GPI). In other words, it could be used with any device requiring either a contact closer or a voltage level change for the interface. These units provide a convenient way to enable computer control of various types of equipment such as security alarms, controlled lighting, and even robotics.

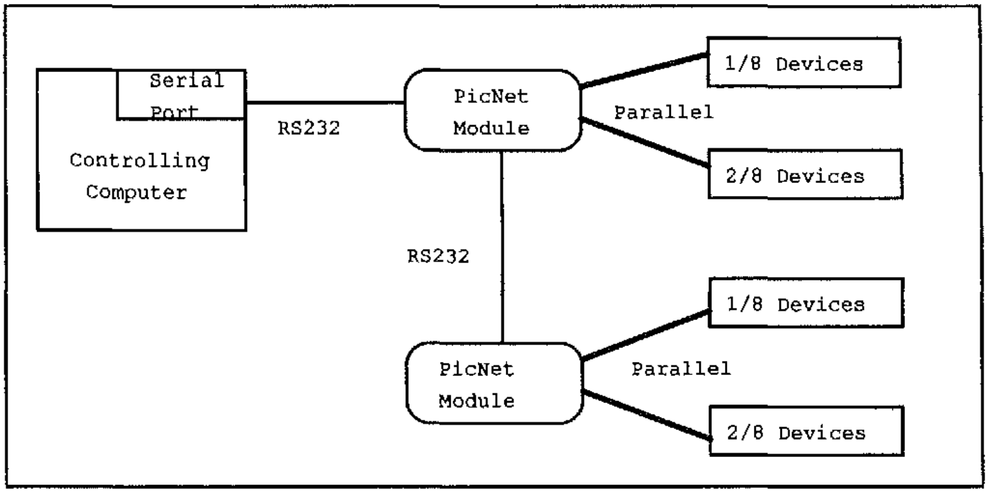


Figure 25: PicNet Module Configuration

The PicNet 8-Port Controller Station is a member of a product line which supports serial-to-parallel input/output (I/O). Other products include a 4-port control and sensing station, a speed controller, and a current detector. Magnetic, infrared, and photocell sensor kits are available for use with the current detector.

The unit works as claimed by the vendor, and customer support is excellent. Any of these products can be obtained directly from the vendor.

Software Interphase, Inc.

82 Cucumber Hill Road
Foster, RI 02825-1212

Email: sinterphas@aol.com

www.sinterphase.com

Tel: (401) 397-2340 Fax: (401) 397-6814

APPENDIX G

Sony Protocol of Remote-1 (9 Pin) Connector

This is the manual which describes the protocol used with the serial remote of the Sony Betacam series. Topology as well as protocol information is given in this text. The document covers the BVW-10, BVW-11, BVW-15, BVW-35, BVW-40, BVW-60, BVW-65, BVW-70, BVW-75, and BVW-96 models; however, the protocol seems to work well with other models. This suggests that there is some consistence between remote control protocols of various Sony tape machines. For instance, I found that this protocol worked well, for most operations, on the BVU-800 U-matic machines. This is one of the units which was loaned to me by WJXT for experimentation purposes during this project.

The documentation is not free or public domain. Sony retains all rights to this information. A copy of this manual, or protocol manuals for other Sony equipment, can be purchased only from the Sony Corporation. The part number of the document used in this thesis is 9-967-137-02.

Sony Broadcast Parts
677 River Oak Parkway
San Jose, CA 95134
Tel: (800) 538-7550

APPENDIX H

MIB Objects Used in this Thesis and the SMICng Include File

```
THESIS-MIB DEFINITIONS ::= BEGIN

-- Title:  UNF Thesis MIB Examples
-- Date:   August 1997
-- By:     Walter Schuller

IMPORTS

    experimental                                FROM RFC1155-SMI

    OBJECT-TYPE                                 FROM RFC-1212;


--******************************************************************
--*   Global Definition of the MIB
--******************************************************************

-- Experimental

      xmit              OBJECT IDENTIFIER ::= { experimental 1 }
      tape              OBJECT IDENTIFIER ::= { experimental 2 }
      switch            OBJECT IDENTIFIER ::= { experimental 3 }


--******************************************************************
--*  Object Definitions Start
--******************************************************************

-- Objects for TRANSMITTER control MIB


txPowerUp OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transmitter power to be
     increased by a predetermined amount."
    ::= { xmit 1 }


txPowerDown OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transmitter power to be
     decreased by a predetermined amount."
    ::= { xmit 2 }
```

```
txExciterA OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes exciter A to be activated."
    ::= { xmit 3 }


txExciterB OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes exciter B to be activated."
    ::= { xmit 4 }


txSysOn OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transmitter high-voltage plates
     to be activated  (transmitter on)."
    ::= { xmit 5 }


txSysOff OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transmitter high-voltage plates
     to be deactivated (transmitter off)."
    ::= { xmit 6 }


txAirA OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes antenna A to be used."
    ::= { xmit 7 }


txAirB OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes antenna B to be used."
    ::= { xmit 8 }
```

```
txAirAB OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes antenna A and B to be used
     (circular polarized)."
    ::= { xmit 9 }


txGenTrans OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes transfer to aux power generator."
    ::= { xmit 10 }


txGenOn OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes activation of aux power generator."
    ::= { xmit 11 }


txGenOff OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes deactivation of aux power generator."
    ::= { xmit 12 }


txVisPwr OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Indication of Visual power output."
    ::= { xmit 13 }


txAurPwr OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Indication of Aural power output."
    ::= { xmit 14 }
```

```
txAPlateKV OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Indication of transmitter A plate KVolts."
    ::= { xmit 15 }


txAPlateAmp OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Indication of transmitter A plate current (Amps)."
    ::= { xmit 16 }


txBPlateKV OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Indication of transmitter B plate KVolts."
    ::= { xmit 17 }


txBPlateAmp OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Indication of transmitter B plate current (Amps)."
    ::= { xmit 18 }

-- ===============================================================


-- Objects for TAPE MACHINE control MIB


sonyPlay OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate PLAY mode."
    ::= { tape 1 }


sonyRecord OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate Record mode."
    ::= { tape 2 }
```

```
sonyForward OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate FORWARD mode."
    ::= { tape 3 }


sonyReverse OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate REVERSE mode."
    ::= { tape 4 }


sonyJogFwd OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate FORWARD-JOG mode."
    ::= { tape 5 }


sonyJogRev OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate REVERSE-JOG mode."
    ::= { tape 6 }


sonyEject OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to activate EJECT mode."
    ::= { tape 7 }


sonyTimecode OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..25))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
    "Timecode of tape in machine at time of request."
    ::= { tape 8 }
```

```
sonyStop OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Sony tape machine to STOP operation."
    ::= { tape 9 }


-- ==============================================================


-- Objects for Audio / Video SWITCH control MIB


switch1 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 1 to activate."
    ::= { switch 1 }


switch2 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 2 to activate."
    ::= { switch 2 }


switch3 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 3 to activate."
    ::= { switch 3 }


switch4 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 4 to activate."
    ::= { switch 4 }


switch5 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 5 to activate."
    ::= { switch 5 }
```

```
switch6 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 6 to activate."
    ::= { switch 6 }


switch7 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 7 to activate."
    ::= { switch 7 }


switch8 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "When set, causes Switch # 8 to activate."
    ::= { switch 8 }


--*******************************************************************
--*  Object Definitions End
--*******************************************************************


END
```

Include File for use with SMICng

```
-- file: thesis.inc
-- Modules referenced by module THESIS-MIB

#condInclude "rfc1155.inc" -- RFC1155-SMI
#condInclude "rfc1212.inc" -- RFC-1212

-- MIB module
#pushOpt
-- Remove strict checking
-- Options:
--    C - check size/range present
--    W - don't allow size/range for items in a sequence
--    7 - restrict INTEGER values below 2G-1
--    R - check (in V1) that INDEX objs are read-only
--    S - require (in V2) that IMPORTS be specified for items in
compliances
--    B - strong checking for size/range of items in index clause
#removeOpt "C W 7 R S B"

-- Loosen checking
-- Options:
--    4 - allow non-standard access for objects
--    K - allow (in v1) zero valued enums
--    O - allow (in v2) hyphens in labels for enumerated values
--    P - allow (in v2) hyphens in descriptors(identifiers)
--    T - no check (in v2) of proper access for items in groups
--    M - no check (in v2) that all NTs and accessible OTs are in a group
--    F - allow integer/integer32 index items without a range
--    G - allow unused IMPORTS and textual conventions
--    N - no check (in v2) of access of objects in notifications
--    I - use (in v1) the v2 rules for checking ACCESS of index items
--#addOpt "4 K O P T M F G N I"
#addOpt "F" -- allow integer/integer32 index items without a range

#condInclude "thesis.mib"
#popOpt
```

APPENDIX I

Example of Other Possible Mibs

and SMICng Include files

```
VIDEOPROC-MIB DEFINITIONS ::= BEGIN

-- Title:          VideoProc MIB
-- Date:           August 1997
-- By:             Walter Schuller
-- Comment:        Example mib for video processing amplifier

IMPORTS
    experimental            FROM RFC1155-SMI

    OBJECT-TYPE             FROM RFC-1212;

--*****************************************************************
--*   Global Definition of the MIB
--*****************************************************************

-- Experimental

      proc              OBJECT IDENTIFIER ::= { experimental 4 }

--*****************************************************************
--*  Object Definitions Start
--*****************************************************************

-- Objects for VIDEOPROC control MIB


videoLevel OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set video level (0=minimum  255=maximum)."
    ::= { proc 1 }


blackLevel OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set black setup level (0=minimum  255=maximum)."
    ::= { proc 2 }
```

```
chromaLevel OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set chroma level (0=minimum  255=maximum)."
    ::= { proc 3 }


huePhase OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set chroma hue (0=maximum conter clockwise  255=minimum
    clockwise)."
    ::= { proc 4 }


burstLevel OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set burst level (0=minimum   255=maximum)."
    ::= { proc 5 }


whiteClip OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set white clip level IRE (0=minimum  255=maximum)."
    ::= { proc 6 }


syncLevel OBJECT-TYPE
    SYNTAX INTEGER(0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Set sync level (0=minimum  255=maximum)."
    ::= { proc 7 }

--*******************************************************************
--*  Object Definitions End
--*******************************************************************


END
```

Include File for SMICng Compiler

```
-- file: videoproc.inc
-- Modules referenced by module VIDEOPROC-MIB

#condInclude "rfc1155.inc" -- RFC1155-SMI
#condInclude "rfc1212.inc" -- RFC-1212

-- MIB module
#pushOpt
-- Remove strict checking
-- Options:
--    C - check size/range present
--    W - don't allow size/range for items in a sequence
--    7 - restrict INTEGER values below 2G-1
--    R - check (in V1) that INDEX objs are read-only
--    S - require (in V2) that IMPORTS be specified for items in
compliances
--    B - strong checking for size/range of items in index clause
#removeOpt "C W 7 R S B"

-- Loosen checking
-- Options:
--    4 - allow non-standard access for objects
--    K - allow (in v1) zero valued enums
--    O - allow (in v2) hyphens in labels for enumerated values
--    P - allow (in v2) hyphens in descriptors(identifiers)
--    T - no check (in v2) of proper access for items in groups
--    M - no check (in v2) that all NTs and accessible OTs are in a group
--    F - allow integer/integer32 index items without a range
--    G - allow unused IMPORTS and textual conventions
--    N - no check (in v2) of access of objects in notifications
--    I - use (in v1) the v2 rules for checking ACCESS of index items
--#addOpt "4 K O P T M F G N I"
#addOpt "F" -- allow integer/integer32 index items without a range

#condInclude "videoproc.mib"
#popOpt
```

Example MIB

```
SCOPE-MIB DEFINITIONS ::= BEGIN

-- Title:         Waveform monitor MIB
-- Date:          August 1997
-- By:            Walter Schuller
-- Comment:       Example possible mib for a waveform monitor

IMPORTS
    experimental         FROM RFC1155-SMI

    OBJECT-TYPE          FROM RFC-1212;

--******************************************************************
--*   Global Definition of the MIB
--******************************************************************

-- Experimental

      scope             OBJECT IDENTIFIER ::= { experimental 5 }

--******************************************************************
--*  Object Definitions Start
--******************************************************************

-- Objects for WAVEFORM MONITOR control MIB


scopeMode OBJECT-TYPE
    SYNTAX INTEGER {
        vector(1),
      wafeform(2)
                   }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Scope type."
    ::= { scope 1 }


waveform OBJECT-TYPE
    SYNTAX INTEGER {
        horz1(1),
        horz2(2),
        line(3),
      vert(4)
                   }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Type of waveform displayed."
    ::= { scope 2 }
```

```
fields OBJECT-TYPE
    SYNTAX INTEGER {
        even(1),
        odd(2)
                   }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Which field to display."
    ::= { scope 3 }


freqResponse OBJECT-TYPE
    SYNTAX INTEGER {
        flat(1),
      lowPass(2)
                   }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Frequency response."
    ::= { scope 4 }


scanLine OBJECT-TYPE
    SYNTAX INTEGER (0..255)
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
    "Scan line displayed in single line mode."
    ::= { scope 5 }


--******************************************************************
--*  Object Definitions End
--******************************************************************


END
```

SMICng Include File

```
-- file: videoproc.inc
-- Modules referenced by module SCOPE-MIB

#condInclude "rfc1155.inc" -- RFC1155-SMI
#condInclude "rfc1212.inc" -- RFC-1212

-- MIB module
#pushOpt
-- Remove strict checking
-- Options:
--    C - check size/range present
--    W - don't allow size/range for items in a sequence
--    7 - restrict INTEGER values below 2G-1
--    R - check (in V1) that INDEX objs are read-only
--    S - require (in V2) that IMPORTS be specified for items in
compliances
--    B - strong checking for size/range of items in index clause
#removeOpt "C W 7 R S B"

-- Loosen checking
-- Options:
--    4 - allow non-standard access for objects
--    K - allow (in v1) zero valued enums
--    O - allow (in v2) hyphens in labels for enumerated values
--    P - allow (in v2) hyphens in descriptors(identifiers)
--    T - no check (in v2) of proper access for items in groups
--    M - no check (in v2) that all NTs and accessible OTs are in a group
--    F - allow integer/integer32 index items without a range
--    G - allow unused IMPORTS and textual conventions
--    N - no check (in v2) of access of objects in notifications
--    I - use (in v1) the v2 rules for checking ACCESS of index items
--#addOpt "4 K O P T M F G N I"
#addOpt "F" -- allow integer/integer32 index items without a range

#condInclude "scope.mib"
#popOpt
```

APPENDIX J

Software Listings

Software listings for one manager and proxy agent is provided for the reader. Other listings can be obtained from the included media.

Manager Program for Sony Tape Machine

```
// GENERAL.H
//
// Walter Schuller
// University of North Florida
//
// 1997

#ifndef GENERAL_H
#define GENERAL_H

#include <stdlib.h>
#include <string.h>
#include "classes.hpp"



typedef struct tagAPPVARS
{
   char*            szAppName;      // name of application
   HINSTANCE        hinstCtl3d;     // 3D control module
   HFONT            hfontNormal;    // font with normal weight
   CSnmpManager*    pSnmpManager;   // our SNMP agent object
} APPVARS, *PAPPVARS, FAR *LPAPPVARS;


typedef enum SNMPMSTATE
   {
   SNMPM_STATE_BROADCAST,
   SNMPM_STATE_QUERYHOST,
   SNMPM_STATE_ADDRTRANS,
   SNMPM_STATE_SETOBJECT
   };


#define GWL_LPAPPVARS DLGWINDOWEXTRA
#define MAINWNDEXTRA DLGWINDOWEXTRA + sizeof(LPAPPVARS)


// Port 161 is the Well-Known Service for SNMP
#define IPPORT_SNMP 161

#endif

//*************************************************************
// END OF FILE                        WALT SCHULLER
// SNMPMP.CPP
// Declaration of the Management Windows Program
//
// Walter Schuller
// University of North Florida
//
// 1997
//


#include "general.hpp"
#include "calls.hpp"


//*************************************************************
// Center Window

void CenterWindow(HWND hwnd)
   {
   RECT rc;
   int cx, cy;

   GetWindowRect(hwnd, &rc);
   cx = GetSystemMetrics(SM_CXSCREEN);
   cy = GetSystemMetrics(SM_CYSCREEN);

   MoveWindow(hwnd, (cx - rc.right + rc.left) / 2,
         (cy - rc.bottom + rc.top) / 2,
         rc.right - rc.left, rc.bottom - rc.top, TRUE);
   }



//************************************************************
// Set the background color of the dialog box, buttons controls and
// static controls to light gray.

HBRUSH AnyWnd_OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, int type)
   {
   if (CTLCOLOR_BTN == type || CTLCOLOR_DLG == type ||
      CTLCOLOR_STATIC == type)
      {
      SetBkMode(hdc, TRANSPARENT);
      return GetStockBrush(LTGRAY_BRUSH);
      }

   return NULL;
   }


//****************************************************************
// Initialize the Main Window.

BOOL MainWnd_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
   {
   PAPPVARS pav = (PAPPVARS) lParam;
   LOGFONT logfont;
      LPFNCTL3DSUBCLASSDLGEX lpfnCtl3dSubclassDlgEx;
```

```
        if ((HINSTANCE) HINSTANCE_ERROR != pav->hinstCtl3d)
              {
              lpfnCtl3dSubclassDlgEx = (LPFNCTL3DSUBCLASSDLGEX)
              GetProcAddress(pav->hinstCtl3d, "Ctl3dSubclassDlgEx");
              if (lpfnCtl3dSubclassDlgEx)
                    (*lpfnCtl3dSubclassDlgEx)(hwnd, 0xffff);
              }

     CenterWindow(hwnd);

     SetWindowLong(hwnd, GWL_LPAPPVARS, (LONG) (LPAPPVARS) pav);

     pav->pSnmpManager = new CSnmpManager(GetWindowInstance(hwnd),  hwnd);

     pav->hfontNormal = (HFONT) SendMessage(GetDlgItem(hwnd,
           IDC_SNMPMDESC), WM_GETFONT, 0, 0);

     GetObject(pav->hfontNormal, sizeof(LOGFONT), (LPSTR) &logfont);

     logfont.lfWeight = FW_NORMAL;

     pav->hfontNormal = CreateFontIndirect(&logfont);

     SetDlgItemText(hwnd, IDC_STAT, "Port CLOSED");


     // Attempt to Open a UDP Port.
     pav->pSnmpManager->Connect(NULL, PT_NOFLAGS, NULL, 0);

     return FALSE;
     }

//****************************************************************
// Process WM_COMMAND Messages for the Main Window.

void MainWnd_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
     {

     HINSTANCE hinst;

     LPSTR lpszObjectId;
     LPSTR lpszObjectValue;
     char szObjectId[64];

     SNMP_OBJECT_TYPE fObjectType;

        PAPPVARS pav = (PAPPVARS) GetWindowLong(hwnd, GWL_LPAPPVARS);

        hinst = GetWindowInstance(hwnd);

     switch (id)
        {
        case IDC_ABOUT:
              #ifdef WIN32
              AboutDlg_Do(hwnd, "SNMP C++/32 Class Library");
              #else
              AboutDlg_Do(hwnd, "SNMP C++/16 Class Library");
              #endif
           break;
```

```
case IDC_PLAY:

       fObjectType = SNMP_OCTET_STRING;

      _fstrcpy(szObjectId, GetString(hinst, IDS_SonyPlay));

       lpszObjectId=szObjectId;

       lpszObjectValue = "1";

       pav->pSnmpManager->state = SNMPM_STATE_SETOBJECT;

       pav->pSnmpManager->SendSetRequest(
       pav->pSnmpManager->szRemoteHost, IPPORT_SNMP, "public",
       1,1, (LPSTR FAR*) &lpszObjectId, (LONG FAR*)
       &lpszObjectValue, &fObjectType);

        break;

case IDC_RECORD:

       fObjectType = SNMP_OCTET_STRING;

       fstrcpy(szObjectId, GetString(hinst, IDS_SonyRecord));

       lpszObjectId=szObjectId;

       lpszObjectValue = "1";

       pav->pSnmpManager->state = SNMPM_STATE_SETOBJECT;

       pav->pSnmpManager->SendSetRequest(
       pav->pSnmpManager->szRemoteHost, IPPORT_SNMP, "public",
       1,1, (LPSTR FAR*) &lpszObjectId, (LONG FAR*)
       &lpszObjectValue, &fObjectType);

        break;

case IDC_FORWARD:

       fObjectType = SNMP_OCTET_STRING;

       fstrcpy(szObjectId, GetString(hinst, IDS_SonyForward));

       lpszObjectId=szObjectId;

       lpszObjectValue = "1";

       pav->pSnmpManager->state = SNMPM_STATE_SETOBJECT;

       pav->pSnmpManager->SendSetRequest(
       pav->pSnmpManager->szRemoteHost, IPPORT_SNMP, "public",
       1,1, (LPSTR FAR*) &lpszObjectId, (LONG FAR*)
       &lpszObjectValue, &fObjectType);

        break;

case IDC_REVERSE:

       fObjectType = SNMP_OCTET_STRING;
```

```
        fstrcpy(szObjectId, GetString(hinst, IDS_SonyReverse));

        lpszObjectId=szObjectId;

        lpszObjectValue = "1";

        pav->pSnmpManager->state = SNMPM_STATE_SETOBJECT;

        pav->pSnmpManager->SendSetRequest(
        pav->pSnmpManager->szRemoteHost, IPPORT_SNMP, "public",
        1,1, (LPSTR FAR*) &lpszObjectId, (LONG FAR*)
        &lpszObjectValue, &fObjectType);

         break;

case IDC_STOP:

        fObjectType = SNMP_OCTET_STRING;

        fstrcpy(szObjectId, GetString(hinst, IDS_SonyStop));

        lpszObjectId=szObjectId;

        lpszObjectValue = "1";

        pav->pSnmpManager->state = SNMPM_STATE_SETOBJECT;

        pav->pSnmpManager->SendSetRequest(
        pav->pSnmpManager->szRemoteHost, IPPORT_SNMP, "public",
        1,1, (LPSTR FAR*) &lpszObjectId, (LONG FAR*)
        &lpszObjectValue, &fObjectType);

         break;


case IDC_EJECT:

        fObjectType = SNMP_OCTET_STRING;

        fstrcpy(szObjectId, GetString(hinst, IDS_SonyEject));

        lpszObjectId=szObjectId;

        lpszObjectValue = "1";

        pav->pSnmpManager->state = SNMPM_STATE_SETOBJECT;

        pav->pSnmpManager->SendSetRequest(
        pav->pSnmpManager->szRemoteHost, IPPORT_SNMP, "public",
        1,1, (LPSTR FAR*) &lpszObjectId, (LONG FAR*)
        &lpszObjectValue, &fObjectType);

         break;
case IDC_GETTIME:
case IDC_HOSTLIST:
        {
   typedef struct tagOBJECTVALUE
          {
          LPSTR lpszObject[14];
          char szObject[14][24];
```

```
            } OBJECTVALUE, FAR *LPOBJECTVALUE;

        LPOBJECTVALUE lpObjectValue;

        int i;

        lpObjectValue = (LPOBJECTVALUE) GlobalAllocPtr(GPTR,
           sizeof(OBJECTVALUE));

        for (i = 0; i < 14; ++i)
            lpObjectValue->lpszObject[i]=
                 lpObjectValue->szObject[i][0];

        i = 0;
        _fstrcpy(lpObjectValue->lpszObject[0], GetString(hinst,
           IDS_SonyTimecode));


        ListBox_GetText(hwndCtl, ListBox_GetCurSel(hwndCtl),
                           pav->pSnmpManager->szRemoteHost);

        pav->pSnmpManager->state = SNMPM_STATE_QUERYHOST;

        pav->pSnmpManager->SendGetRequest(
                 pav->pSnmpManager->szRemoteHost, IPPORT_SNMP,
                 "public", 0, 1, (LPSTR FAR*) lpObjectValue);

        MessageBeep(-10);

        GlobalFreePtr(lpObjectValue);

           }
       break;


// Broadcast a Message to Locate Agents

      case IDC_QUERYALLHOSTS:
         {
         LPCSTR lpObject;

         char szObjectId[] = "1.3.6.1.2.1.1.1.0";
         lpObject = (LPCSTR) szObjectId;

         pav->pSnmpManager->state = SNMPM_STATE_BROADCAST;

         ListBox_ResetContent(GetDlgItem(hwnd, IDC_HOSTLIST));

         pav->pSnmpManager->SendGetRequest("255.255.255.255",
            IPPORT_SNMP, "public", 1, 1, (LPSTR FAR*) &lpObject);
         break;
         }
      }
   }
```

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

void MainWnd_OnDestroy(HWND hwnd)
   {
   PAPPVARS pav = (PAPPVARS) GetWindowLong(hwnd, GWL_LPAPPVARS);

   if (PT_CLOSED != pav->pSnmpManager->State())
      pav->pSnmpManager->Close(TRUE);

   delete pav->pSnmpManager;

   PostQuitMessage(0);
   }

//****************************************************************
// Process Main Window Messages

LRESULT CALLBACK MainWnd_WndProc(HWND hwnd, UINT msg, WPARAM wParam,
      LPARAM lParam)
   {
   switch(msg)
      {
      HANDLE_MSG(hwnd, WM_INITDIALOG, MainWnd_OnInitDialog);
      HANDLE_MSG(hwnd, WM_COMMAND, MainWnd_OnCommand);
      HANDLE_MSG(hwnd, WM_DESTROY, MainWnd_OnDestroy);

            #ifdef WIN32
            HANDLE_MSG(hwnd, WM_CTLCOLORBTN, AnyWnd_OnCtlColor);
            HANDLE_MSG(hwnd, WM_CTLCOLORDLG, AnyWnd_OnCtlColor);
            HANDLE_MSG(hwnd, WM_CTLCOLORSTATIC, AnyWnd_OnCtlColor);
            #else
            HANDLE_MSG(hwnd, WM_CTLCOLOR, AnyWnd_OnCtlColor);
            #endif
      }

   return DefWindowProc(hwnd, msg, wParam, lParam);
   }

//****************************************************************
// Initialize Application and Establish Message Loop.

int PASCAL WinMain(HINSTANCE hinstCurrent, HINSTANCE hinstPrevious,
      LPSTR lpszCmdLine, int nCmdShow)
   {
   WNDCLASS wndclass;
   HWND hwnd;
   MSG msg;
   APPVARS av;
   LPFNCTL3DREGISTER lpfnCtl3dRegister;
   LPFNCTL3DAUTOSUBCLASS lpfnCtl3dAutoSubclass;

   av.szAppName = "SNMPM";

      #ifdef WIN32
            av.hinstCtl3d = LoadLibrary("CTL3D32.DLL");
      #else
            av.hinstCtl3d = LoadLibrary("CTL3DV2.DLL");
            if (av.hinstCtl3d < (HINSTANCE) HINSTANCE_ERROR)
                  av.hinstCtl3d = LoadLibrary("CTL3D.DLL");
      #endif
      if (av.hinstCtl3d >= (HINSTANCE) HINSTANCE_ERROR)
            {
            lpfnCtl3dRegister = (LPFNCTL3DREGISTER)
            GetProcAddress(av.hinstCtl3d, "Ctl3dRegister");

            if(lpfnCtl3dRegister)
                  (*lpfnCtl3dRegister)(hinstCurrent);

            lpfnCtl3dAutoSubclass = (LPFNCTL3DAUTOSUBCLASS)
            GetProcAddress(av.hinstCtl3d, "Ctl3dAutoSubclass");

            if (lpfnCtl3dAutoSubclass)
                  (*lpfnCtl3dAutoSubclass)(hinstCurrent);
            }

// Register Main Window Class
   if (!hinstPrevious)
      {
      wndclass.style            = CS_HREDRAW | CS_VREDRAW;
      wndclass.lpfnWndProc      = MainWnd_WndProc;
      wndclass.cbClsExtra       = 0;
      wndclass.cbWndExtra       = MAINWNDEXTRA;
      wndclass.hInstance        = hinstCurrent;
      wndclass.hIcon            = LoadIcon(hinstCurrent, "MAINICON");
      wndclass.hCursor          = LoadCursor(NULL, IDC_ARROW);
      wndclass.hbrBackground    = GetStockBrush(LTGRAY_BRUSH);
      wndclass.lpszMenuName     = NULL;
      wndclass.lpszClassName    = av.szAppName;

      RegisterClass(&wndclass);
      }

// Create the Main Window.
   hwnd = CreateDialogParam(hinstCurrent, av.szAppName, 0,
         (DLGPROC) MainWnd_WndProc, (LPARAM) (LPAPPVARS) &av);
   ShowWindow(hwnd, nCmdShow);

// Handle Messages
   while (GetMessage(&msg, NULL, 0, 0))
      if (!IsDialogMessage(hwnd, &msg))
         {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
         }
      if (av.hinstCtl3d >= (HINSTANCE) HINSTANCE_ERROR)
            {
            LPFNCTL3DUNREGISTER lpfnCtl3dUnregister;
            lpfnCtl3dUnregister = (LPFNCTL3DREGISTER)
                  GetProcAddress(av.hinstCtl3d, "Ctl3dUnregister");
            if (lpfnCtl3dUnregister)
                  (*lpfnCtl3dUnregister)(hinstCurrent);
            FreeLibrary(av.hinstCtl3d);
            }
   return msg.wParam;
   }

//**********************************************************
//  END OF FILE                              WALT SCHULLER
// CLASSES.H
//
// Walter Schuller
// University of North Florida
//
// 1997

#ifndef CLASSES_H
#define CLASSES_H

#include "\Powertcp\include\common.h"
#include "\Powertcp\include\powersnm.hpp"
#include "snmpm.hh"



// SNMP Class obtained from Dart Communications sample

class CSnmpManager : public CPowerSnmp
{
public:

   CSnmpManager(HINSTANCE hinst, HWND hwnd);

   HWND        hwndMain;            // identifies the main window
   HWND        hwndAddrTran;        // address translation table
   int         nAttList;            // current list in our AT table
   int         state;               // type of query being made
   char        szRemoteHost[64];    // name of remote host
   char        szLastObjectId[64];  // last object id queired

protected:

   void ConnectSnmpEvent(LPCSTR lpszLocalDotAddr, WORD wLocalPort,
         LPCSTR lpszLocalName, WORD wMaxByteCnt);

   void RecvSnmpEvent(LPCSTR lpszCommunity, DWORD dwRequestId,
         SNMP_ERROR codeSnmpError, int nErrorIndex, SNMP_TYPE,
      MessageType,
         UINT nObjects, LPSTR FAR* lpszObjectId, LONG FAR*
      lpObjectValue,
         SNMP_OBJECT_TYPE FAR* lpObjectType, LPCSTR   lpszRemoteDotAddr,
         WORD wRemotePort);

   void ExceptionEvent(PT_EXCEPTION codeError, LPCSTR lpszErrorDesc);
};


#endif

//*****************************************************************
//  END OF FILE                                WALT SCHULLER
// Cmanager.cpp
// Declaration of the Management Class Functions
// SNMP virtual class functions from Dart Communications samples
//
//
// Walter Schuller
// University of North Florida
//
// 1997
//
//*****************************************************************

#include "general.hpp"
#include "calls.hpp"

//*****************************************************************


CSnmpManager::CSnmpManager(HINSTANCE hinst, HWND hwnd) :
      CPowerSnmp(hinst), hwndMain(hwnd)
   {

   }


//*****************************************************************
// A UDP Port has been Successfully Allocated

void CSnmpManager::ConnectSnmpEvent(LPCSTR lpszLocalDotAddr, WORD
      wLocalPort, LPCSTR lpszLocalName, WORD wMaxByteCnt)
   {
   SetDlgItemText(hwndMain, IDC_STAT, " UDP Port Opened");
   }


//**************************************************************
// Process SNMP Messages Received


void CSnmpManager::RecvSnmpEvent(LPCSTR lpszCommunity, DWORD
      dwRequestId,  SNMP_ERROR codeSnmpError, int nErrorIndex,
      SNMP_TYPE MessageType,  UINT nObjects, LPSTR FAR* lpszObjectId,
      LONG FAR* lpObjectValue, SNMP_OBJECT_TYPE FAR* lpObjectType,
      LPCSTR lpszRemoteDotAddr, WORD wRemotePort)
   {

   switch (state)
      {
      case SNMPM_STATE_BROADCAST:

         ListBox_AddString(GetDlgItem(hwndMain,
               IDC_HOSTLIST), lpszRemoteDotAddr);
      break;

      case SNMPM_STATE_QUERYHOST:
         {

         HINSTANCE hinst;
         UINT i;
         hinst = GetWindowInstance(hwndMain);

         for (i = 0; i < nObjects; ++i)
            {
            if (0 == _fstrcmp(GetString(hinst, IDS_SonyTimecode),
                  lpszObjectId[i]))
               SetDlgItemText(hwndMain,
                     IDC_TIMECODE, (LPSTR) lpObjectValue[i]+2);

            }

         break;
         }


      default:
         break;
      }
   }

//**************************************************************
// Display Error

void CSnmpManager::ExceptionEvent(PT_EXCEPTION codeError,
      LPCSTR lpszErrorDesc)
   {
   SetDlgItemText(hwndMain, IDC_STAT, lpszErrorDesc);
   }


//****************************************************************
// END OF FILE                                WALT SCHULLER
```

Agent Program for Sony Tape Machine

```
// GENERAL.H
//
// Walter Schuller
// University of North Florida
//
// 1997

#ifndef GENERAL_H
#define GENERAL_H

#include <stdlib.h>
#include <string.h>

#include "classes.hpp"



// The APPVARS structure contains information used the SNMP Classes

typedef struct tagAPPVARS
{
   char*            szAppName;               // name of application
   HINSTANCE        hinstCtl3d;              // 3D control module
   HFONT            hfontNormal;             // font with normal weight
   CSnmpAgent*      pSnmpAgent;              // our SNMP agent object
} APPVARS, *PAPPVARS, FAR *LPAPPVARS;


#define GWL_LPAPPVARS DLGWINDOWEXTRA
#define MAINWNDEXTRA DLGWINDOWEXTRA + sizeof(LPAPPVARS)


// Port 161 is the well-known service for SNMP
#define IPPORT_SNMP 161



#endif

//***************************************************************
//  END OF FILE                          WALT SCHULLLER
// SNMPAP.CPP
// Declaration of the Agent Windows Program
//
// Walter Schuller
// University of North Florida
//
// 1997


#include "general.hpp"
#include "calls.hpp"



//**************************************************
// Center Window

void CenterWindow(HWND hwnd)
   {
   RECT rc;
   int cx, cy;

   GetWindowRect(hwnd, &rc);
   cx = GetSystemMetrics(SM_CXSCREEN);
   cy = GetSystemMetrics(SM_CYSCREEN);

   MoveWindow(hwnd, (cx - rc.right + rc.left) / 2,
         (cy - rc.bottom + rc.top) / 2,
         rc.right - rc.left, rc.bottom - rc.top, TRUE);
   }


//*****************************************************************
// Initialize Main Window.

BOOL MainWnd_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
   {
   PAPPVARS pav = (PAPPVARS) lParam;
   LOGFONT logfont;
   LPFNCTL3DSUBCLASSDLGEX lpfnCtl3dSubclassDlgEx;
   HINSTANCE hInst = GetWindowInstance(hwnd);

      // Enable dialog to use 3D controls.
      if ((HINSTANCE) HINSTANCE_ERROR != pav->hinstCtl3d)
            {
            lpfnCtl3dSubclassDlgEx = (LPFNCTL3DSUBCLASSDLGEX)
            GetProcAddress(pav->hinstCtl3d, "Ctl3dSubclassDlgEx");
            if (lpfnCtl3dSubclassDlgEx)
                  (*lpfnCtl3dSubclassDlgEx)(hwnd, 0xffff);
            }

   CenterWindow(hwnd);

   SetWindowLong(hwnd, GWL_LPAPPVARS, (LONG) (LPAPPVARS) pav);

   pav->hfontNormal = (HFONT) SendMessage(GetDlgItem(hwnd,
         IDC_STATUS), WM_GETFONT, 0, 0);
   GetObject(pav->hfontNormal, sizeof(LOGFONT), (LPSTR) &logfont);

   logfont.lfWeight = FW_NORMAL;

   pav->hfontNormal = CreateFontIndirect(&logfont);

   SendDlgItemMessage(hwnd, IDC_STATUS, WM_SETFONT,
         (WPARAM) pav->hfontNormal, 0);

   SetDlgItemText(hwnd, IDC_CMDRESPONSE, "NONE");
   SetDlgItemText(hwnd, IDC_CMDEXECUTED, "NONE");
   SetDlgItemText(hwnd, IDC_CMDRECEIVED, "NONE");

   pav->pSnmpAgent = new CSnmpAgent(GetWindowInstance(hwnd), hwnd);

   pav->pSnmpAgent->Connect(NULL, PT_NOFLAGS, NULL, IPPORT_SNMP);

   return FALSE;
   }

//*****************************************************
// Process WM_COMMAND Messages from Main Window.

void MainWnd_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
   {
   switch (id)
      {

      case IDC_TESTBUTTON:
             {
// Test the Datacomm Port

                  Ccontrol ctl;

                  ctl.Sony_forward();
                  delay(10);
                  ctl.Sony_rewind();
                  delay(5);
                  ctl.Sony_play();
                  delay(5);
                  ctl.Sony_time();
                  SetDlgItemText(hwnd, IDC_TESTWINDOW, ctl.RxBuf);
                  delay(3);
//                ctl.Sony_record();
//                delay(5);
                  ctl.Sony_stop();


        break;
             }

//----------------------------------------------------------------------

      case IDC_ABOUT:
             {
// Display the ABOUT Dialog Box

                  AboutDlg_Do(hwnd, "SNMP C++/16 Class Library");
```

```
            break;
              }

        }

    }


//****************************************************************
// Post a WM_QUIT Message and Set the Exit Code to 0.

void MainWnd_OnDestroy(HWND hwnd)
    {
    PAPPVARS pav = (PAPPVARS) GetWindowLong(hwnd, GWL_LPAPPVARS);

    DeleteObject(pav->hfontNormal);

    if (PT_CLOSED != pav->pSnmpAgent->State())
        pav->pSnmpAgent->Close(TRUE);

    delete pav->pSnmpAgent;

    PostQuitMessage(0);
    }


//*************************************************************
// Set the background color of buttons and static controls to light
gray.

HBRUSH MainWnd_OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, int type)
    {
    if (CTLCOLOR_BTN == type || CTLCOLOR_STATIC == type)
        {
        SetBkMode(hdc, TRANSPARENT);
        return GetStockBrush(LTGRAY_BRUSH);
        }

    return NULL;
    }

//*************************************************************
// Process Main Window Messages

LRESULT CALLBACK MainWnd_WndProc(HWND hwnd, UINT msg, WPARAM wParam,
        LPARAM lParam)
    {
    switch(msg)
        {
        HANDLE_MSG(hwnd, WM_INITDIALOG, MainWnd_OnInitDialog);
        HANDLE_MSG(hwnd, WM_COMMAND, MainWnd_OnCommand);
        HANDLE_MSG(hwnd, WM_DESTROY, MainWnd_OnDestroy);

                #ifdef WIN32
                HANDLE_MSG(hwnd, WM_CTLCOLORBTN, MainWnd_OnCtlColor);
                HANDLE_MSG(hwnd, WM_CTLCOLORSTATIC, MainWnd_OnCtlColor);
                #else
                HANDLE_MSG(hwnd, WM_CTLCOLOR, MainWnd_OnCtlColor);
                #endif
        }
```

```
   return DefWindowProc(hwnd, msg, wParam, lParam);
   }


//*************************************************************
// Initialize Application and Establish Message Loop.

int PASCAL WinMain(HINSTANCE hinstCurrent, HINSTANCE hinstPrevious,
      LPSTR lpszCmdLine, int nCmdShow)
   {
   WNDCLASS wndclass;
   HWND hwnd;
   MSG msg;
   APPVARS av;
   LPFNCTL3DREGISTER lpfnCtl3dRegister;
   LPFNCTL3DAUTOSUBCLASS lpfnCtl3dAutoSubclass;

   av.szAppName = "SNMPA";

      #ifdef WIN32
            av.hinstCtl3d = LoadLibrary("CTL3D32.DLL");
      #else
            av.hinstCtl3d = LoadLibrary("CTL3DV2.DLL");
            if (av.hinstCtl3d < (HINSTANCE) HINSTANCE_ERROR)
                  av.hinstCtl3d = LoadLibrary("CTL3D.DLL");
      #endif

      if (av.hinstCtl3d >= (HINSTANCE) HINSTANCE_ERROR)
            {
            lpfnCtl3dRegister = (LPFNCTL3DREGISTER)
                  GetProcAddress(av.hinstCtl3d, "Ctl3dRegister");

            if (lpfnCtl3dRegister)
                  (*lpfnCtl3dRegister)(hinstCurrent);

            lpfnCtl3dAutoSubclass = (LPFNCTL3DAUTOSUBCLASS)
                  GetProcAddress(av.hinstCtl3d, "Ctl3dAutoSubclass");

            if (lpfnCtl3dAutoSubclass)
                  (*lpfnCtl3dAutoSubclass)(hinstCurrent);
            }

// Register Main Window Class
   if (!hinstPrevious)
      {
      wndclass.style          = CS_HREDRAW | CS_VREDRAW;
      wndclass.lpfnWndProc    = MainWnd_WndProc;
      wndclass.cbClsExtra     = 0;
      wndclass.cbWndExtra     = MAINWNDEXTRA;
      wndclass.hInstance      = hinstCurrent;
      wndclass.hIcon          = LoadIcon(hinstCurrent, "MAINICON");
      wndclass.hCursor        = LoadCursor(NULL, IDC_ARROW);
      wndclass.hbrBackground  = GetStockBrush(LTGRAY_BRUSH);
      wndclass.lpszMenuName   = NULL;
      wndclass.lpszClassName  = av.szAppName;

      RegisterClass(&wndclass);
      }
```

```
// Create Main Window.
   hwnd = CreateDialogParam(hinstCurrent, av.szAppName, 0,
          (DLGPROC) MainWnd_WndProc, (LPARAM) (LPAPPVARS) &av);
   ShowWindow(hwnd, nCmdShow);

// Get and Dispatch Messages
   while (GetMessage(&msg, NULL, 0, 0))
      if (!IsDialogMessage(hwnd, &msg))
         {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
         }

   if (av.hinstCtl3d >= (HINSTANCE) HINSTANCE_ERROR)
            {
            LPFNCTL3DUNREGISTER lpfnCtl3dUnregister;

            lpfnCtl3dUnregister = (LPFNCTL3DREGISTER)
                  GetProcAddress(av.hinstCtl3d, "Ctl3dUnregister");
            if (lpfnCtl3dUnregister)
                  (*lpfnCtl3dUnregister)(hinstCurrent);
            FreeLibrary(av.hinstCtl3d);
            }

   return msg.wParam;
   }

//****************************************************************
// END OF FILE                               WALT SCHULLER
// CLASSES.HPP
// Classes defined
//
// Walter Schuller
// University of North Florida
//
// 1997
//


#ifndef CLASSES_H
#define CLASSES_H

#include "\Powertcp\include\common.h"
#include "\Powertcp\include\powersnm.hpp"
#include "snmpa.hh"
#include "message.h"
#include "wsc.h"
#include "ascii.h"


//***************************************************************
//  SNMP Agent Class


class CSnmpAgent : public CPowerSnmp
{
public:

   CSnmpAgent(HINSTANCE hinst, HWND hwnd);

   HWND        hwndMain;


   char         szLocalDotAddr[64];
   char         szSysDescr[64];
   char         szSysObjectId[32];
   char         szSysContact[64];
   char         szSysName[64];
   char         szSysLocation[64];

protected:

   void ConnectSnmpEvent(LPCSTR lpszLocalDotAddr, WORD wLocalPort,
         LPCSTR lpszLocalName, WORD wMaxByteCnt);

   void RecvSnmpEvent(LPCSTR lpszCommunity, DWORD dwRequestId,
         SNMP_ERROR codeSnmpError, int nErrorIndex, SNMP_TYPE
      MessageType, UINT nObjects, LPSTR FAR* lpszObjectId, LONG FAR*
      lpObjectValue, SNMP_OBJECT_TYPE FAR* lpObjectType, LPCSTR
      lpszRemoteDotAddr, WORD wRemotePort);

   void ExceptionEvent(PT_EXCEPTION codeError, LPCSTR lpszErrorDesc);
};
```

```
// ***************************************************************
// Ccontrol Class
// Class made from the MarshallSoft toolkit

      class Ccontrol
            {

            public:
            char RxBuf[128];

            public:
                  Ccontrol(int = COM1, int = Baud38400, int =
                  OddParity, int = OneStopBit, int = WordLength8);

                  ~Ccontrol();

                  void Sony_play();
                  void Sony_stop();
                  void Sony_rewind();
                  void Sony_forward();
                  void Sony_eject();
                  void Sony_record();
                  void Sony_time();

            private:
                  int Port;
                  int BaudCode;
                  int Parity;
                  int StopBits;
                  int DataBits;

                  char time_buffer[100];

            int send_cmd(int *cmdptr);

       };

#endif

//*****************************************************************
//  END OF FILE                                  WALT SCHULLER
// Ccontrol.cpp
//
// Class made from the MarshallSoft toolkit
// Used to control Sony Tape Machines
//
// Walter Schuller
// University of North Florida
//
// 1997


#include "general.hpp"
#include "calls.hpp"

// Sony Tape Control Command Strings

 int sony_who[]          =  {3,0x00,0x11,0x11};
 int sony_enable[]       =  {3,0x00,0x1d,0x1d};
 int sony_play[]         =  {3,0x20,0x01,0x21};
 int sony_time[]         =  {4,0x61,0x0c,0x01,0x6e};
 int sony_stop[]         =  {3,0x20,0x00,0x20};
 int sony_rewind[]       =  {3,0x20,0x20,0x40};
 int sony_forward[]      =  {3,0x20,0x10,0x30};
 int sony_eject[]        =  {3,0x20,0x0f,0x2f};
 int sony_stdby_on[]     =  {3,0x20,0x05,0x25};
 int sony_stdby_off[]    =  {3,0x20,0x04,0x24};
 int sony_record[]       =  {3,0x20,0x02,0x22};
 int sony_status[]       =  {4,0x61,0x20,0x2a,0x08};
 int sony_auto_edit[]    =  {3,0x20,0x42,0x62};
 int sony_edit_on[]      =  {3,0x20,0x65,0x85};
 int sony_edit_off[]     =  {3,0x20,0x64,0x84};
 int sony_in_entry[]     =  {3,0x40,0x10,0x50};
 int sony_tab_plus[]     =  {3,0x40,0x18,0x58};
 int sony_in_reset[]     =  {3,0x40,0x20,0x60};
 int sony_out_reset[]    =  {3,0x40,0x21,0x61};
 int sony_set_edit[]     =  {4,0x41,0x30,0x50,0xc1};
 int sony_ain_reset[]    =  {3,0x40,0x22,0x62};
 int sony_aout_reset[]   =  {3,0x40,0x23,0x63};
 int sony_reset_edit[]   =  {4,0x41,0x30,0x00,0x71};
 int sony_preroll[]      =  {3,0x20,0x30,0x50};
 int sony_auto_on[]      =  {3,0x40,0x41,0x81};
 int sony_auto_off[]     =  {3,0x40,0x40,0x80};

//*****************************************************************
// Constructor

Ccontrol::Ccontrol(int v, int w, int x, int y, int z)
      {
      Port=v;
      BaudCode=w;
      Parity=x;
      StopBits=y;
      DataBits=z;

      SioReset(Port, 128, 128);
      SioRxClear(Port);
      SioParms(Port, Parity, StopBits, DataBits);
      SioBaud(Port,BaudCode);
      }
```

```
//*****************************************************************
// Destructor

Ccontrol::~Ccontrol()
      {
      SioDone(Port);
      MessageBeep(-10);
      }


//*****************************************************************
// Part of Send Command

 int Ccontrol::send_cmd(int *cmdptr)
 {
       int i=0, num, count = 0;
       num = *cmdptr;
       for (count=1;count<num+1;count++)
            {
//          delay(0.1);
            SioPutc(Port,*(cmdptr + count));
            }

       return 0;
 }

//*****************************************************************
// SONY PLAY Command

void Ccontrol::Sony_play()
      {
      send_cmd(sony_play);
      }


//*****************************************************************
// SONY STOP Command

void Ccontrol::Sony_stop()
      {
      send_cmd(sony_stop);
      }



//*****************************************************************
// SONY REWIND Command

void Ccontrol::Sony_rewind()
      {
      send_cmd(sony_rewind);
      }
```

```
//******************************************************************
// SONY FORWARD Command

void Ccontrol::Sony_forward()
      {
      send_cmd(sony_forward);
      }


//******************************************************************
// SONY EJECT Command

void Ccontrol::Sony_eject()
      {
      send_cmd(sony_eject);
      }


//******************************************************************
// SONY RECORD Command

void Ccontrol::Sony_record()
      {
      send_cmd(sony_record);
      }


//******************************************************************
// SONY GET TIMECODE Command

//     returned data
//    ---------------
// 74 04 52 15 1 0 f0  version requires the modification of frame field
//       F  S  M F      drop frame
//     -40H

// 74 04 12 15 1 0 f0  other version requires no modification of frame
field
//       F  S  M F      non-drop frame

// assume drop frame tape format


void Ccontrol::Sony_time()
      {
      int i, count=0, drop_frame;
      char hours[20], mins[20], secs[20], frams[20];

      fstrcpy(time_buffer,"                             ");

      SioRxClear(Port);                   /* clear port */

      send_cmd(sony_time);

      while(SioRxQue(Port)<5) {};      /* wait for return message */


        do
              {                      /* get chars from buffer */
              delay(0.3);
              i = SioGetc(Port);
              if (i > -1) *(time_buffer+count) = i;
              count++;
              }
        while(i > -1);

        *(time_buffer+count) = '\0';

        if (*(time_buffer+2)&0x40)
                  {
                  *(time_buffer+2)-=0x40;     /* correct for
                                                dropframe */

                  drop_frame=TRUE;
                  }
        else
                  drop_frame=FALSE;

        count=2;

//     convert time to integer and write results to RxBuf

         do
              {
                  if      (*(time_buffer+count) > 0x09 &&
*(time_buffer+count) < 0x20) *(time_buffer+count)-=6;
                  else if (*(time_buffer+count) > 0x19 &&
*(time_buffer+count) < 0x30) *(time_buffer+count)-=12;
                  else if (*(time_buffer+count) > 0x29 &&
*(time_buffer+count) < 0x40) *(time_buffer+count)-=18;
                  else if (*(time_buffer+count) > 0x39 &&
*(time_buffer+count) < 0x50) *(time_buffer+count)-=24;
                  else if (*(time_buffer+count) > 0x49 &&
*(time_buffer+count) < 0x60) *(time_buffer+count)-=30;
                  else if (*(time_buffer+count) > 0x59)
*(time_buffer+count)=60;
                  count++;
              }
         while (count < 6);

       _itoa(*(time_buffer+5), hours, 10);
       _itoa(*(time_buffer+4), mins,  10);
       _itoa(*(time_buffer+3), secs,  10);
       _itoa(*(time_buffer+2), frams, 10);

         _fstrcpy(RxBuf,"  TimeCode = ");
         _fstrcat(RxBuf,hours);
         _fstrcat(RxBuf,":");
         _fstrcat(RxBuf,mins);
         _fstrcat(RxBuf,":");
         _fstrcat(RxBuf,secs);
         _fstrcat(RxBuf,":");
         _fstrcat(RxBuf,frams);
       };


//*****************************************************************
// END OF FILE                        WALT SCHULLER
// Cagent.cpp
//
// Declaration of the Agent Class Functions
// SNMP virtual class functions from Dart Communications
//
// Walter Schuller
// University of North Florida
//
// 1997
//
//


//**************************************************************

#include "general.hpp"
#include "calls.hpp"

//**************************************************************


CSnmpAgent::CSnmpAgent(HINSTANCE hinst, HWND hwnd) :
      CPowerSnmp(hinst), hwndMain(hwnd)
   {

   }


//****************************************************************
// A UDP port has been successfully allocated.

void CSnmpAgent::ConnectSnmpEvent(LPCSTR lpszLocalDotAddr, WORD
wLocalPort, LPCSTR lpszLocalName, WORD wMaxByteCnt)
   {
   char szStatus[128];

   _fstrcpy(szLocalDotAddr, lpszLocalDotAddr);
   wsprintf(szStatus, "Listening on port %d.  MaxPacketSize = %d",
         wLocalPort, wMaxByteCnt);

   SetDlgItemText(hwndMain, IDC_STATUS, szStatus);
   }


//**************************************************************
// Process Received SNMP Messages.

void CSnmpAgent::RecvSnmpEvent(LPCSTR lpszCommunity, DWORD  dwRequestId,
SNMP_ERROR codeSnmpError, int nErrorIndex,      SNMP_TYPE MessageType,
UINT nObjects, LPSTR FAR* lpObjectId,     LONG FAR* lpObjectValue,
SNMP_OBJECT_TYPE FAR* lpObjectType,       LPCSTR lpszRemoteDotAddr, WORD
wRemotePort)
   {
   SNMP_OBJECT_TYPE fObjectType = SNMP_OCTET_STRING;

   UINT i;
   HINSTANCE hinst = GetWindowInstance(hwndMain);
   Ccontrol      ctl;

   char     *ptrsz;

   codeSnmpError = SNMP_NO_ERROR;

//-------------------------------------------------------
// SNMP GET REQUEST Type Message

   switch (MessageType)
      {

      case SNMP_GET_REQUEST:

         lpObjectValue = (LONG FAR*) GlobalAllocPtr(GPTR, nObjects *
            sizeof(LONG));
         lpObjectType = (SNMP_OBJECT_TYPE FAR*)
         GlobalAllocPtr(GPTR, nObjects * sizeof(SNMP_OBJECT_TYPE
            FAR*));


         for (i = 0; i < nObjects; ++i)
            {
            if (IsEqual(hinst, IDS_SonyTimecode, lpObjectId[i]))
               {

              SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "GET
                  TIMECODE");
              SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "SEND
                  TIMECODE");

               ctl.Sony_time();
               ptrsz = ctl.RxBuf;

               SetDlgItemText(hwndMain, IDC_CMDRESPONSE, ptrsz);


               lpObjectValue[i] = GetStringProvided(ptrsz);

               lpObjectType[i] = SNMP_OCTET_STRING;



               }

            }

         SendGetResponse(lpszRemoteDotAddr, wRemotePort,
            lpszCommunity,  dwRequestId, codeSnmpError, nErrorIndex,
            nObjects,  lpObjectId, lpObjectValue, lpObjectType);

            GlobalFreePtr(lpObjectValue);
            GlobalFreePtr(lpObjectType);
         break;

//-----------------------------------------------------
//  SNMP SET REQUEST Type Message

       case SNMP_SET_REQUEST:
```

```
for (i = 0; i < nObjects; ++i)
   {
   if (IsEqual(hinst, IDS_SonyPlay, lpObjectId[i]))
         {
         SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "START
         PLAY");
         SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "SET
         PLAY");
         SetDlgItemText(hwndMain, IDC_CMDRESPONSE,
         "COMPLETE");

               ctl.Sony_play();
         }

   else if (IsEqual(hinst, IDS_SonyStop, lpObjectId[i]))
         {
         SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "STOP
         TAPE");
         SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "SET
         STOP");
         SetDlgItemText(hwndMain, IDC_CMDRESPONSE,
         "COMPLETE");

               ctl.Sony_stop();
         }

   else if (IsEqual(hinst, IDS_SonyReverse, lpObjectId[i]))
         {
         SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "START
         TAPE REVERSE");
         SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "SET
         REVERSE");
         SetDlgItemText(hwndMain, IDC_CMDRESPONSE,
         "COMPLETE");

               ctl.Sony_rewind();
         }

   else if (IsEqual(hinst, IDS_SonyForward, lpObjectId[i]))
         {
         SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "START
         TAPE FORWARD");
         SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "SET
         FORWARD");
         SetDlgItemText(hwndMain, IDC_CMDRESPONSE,
         "COMPLETE");

               ctl.Sony_forward();
         }

   else if (IsEqual(hinst, IDS_SonyEject, lpObjectId[i]))
         {
         SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "EJECT
         TAPE");
         SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "SET
         EJECT");
         SetDlgItemText(hwndMain, IDC_CMDRESPONSE,
         "COMPLETE");

               ctl.Sony_eject();
```

```
                  }

            else if (IsEqual(hinst, IDS_SonyRecord, lpObjectId[i]))
                  {
                  SetDlgItemText(hwndMain, IDC_CMDEXECUTED, "START
                  RECORD");
                  SetDlgItemText(hwndMain, IDC_CMDRECEIVED, "SET
                  RECORD");
                  SetDlgItemText(hwndMain, IDC_CMDRESPONSE,
                  "COMPLETE");

                        ctl.Sony_record();
                  }

            }

         SendGetResponse(lpszRemoteDotAddr, wRemotePort,
            lpszCommunity,   dwRequestId, SNMP_NO_ERROR, 0, nObjects,
            lpObjectId, lpObjectValue, lpObjectType);
         break;

      }

   }


//******************************************************************
// Display Error Message

void CSnmpAgent::ExceptionEvent(PT_EXCEPTION codeError,
      LPCSTR lpszErrorDesc)
   {
   SetDlgItemText(hwndMain, IDC_STATUS, lpszErrorDesc);
   }


//*************************************************************
//  END OF FILE                              WALT SCHULLER
```

Common Calls used by Both Programs

```
// CALLS.H
//
// Walter Schuller
// University of North Florida
//
// 1997

#ifndef CALLS_H
#define CALLS_H

#include <time.h>

// prototypes

extern LONG GetLongObject(HWND, int);

extern LPSTR GetString(HINSTANCE, WORD);

extern BOOL IsEqual(HINSTANCE, int, LPSTR);

extern LONG GetStringObject(HWND, int, LPSTR, int);

extern LONG GetStringProvided(LPSTR);

void   delay(long);

void   sleep(clock_t);


// Function prototypes for CTL3D.DLL

typedef BOOL (CALLBACK* LPFNCTL3DREGISTER)(HINSTANCE);

typedef BOOL (CALLBACK* LPFNCTL3DAUTOSUBCLASS)(HINSTANCE);

typedef BOOL (CALLBACK* LPFNCTL3DSUBCLASSDLGEX)(HWND, DWORD);

typedef BOOL (CALLBACK* LPFNCTL3DUNREGISTER)(HINSTANCE);

extern "C" void AboutDlg_Do(HWND hwnd, LPCSTR lpszLibsUsed);

#endif

//*******************************************************************
//  END OF FILE                           WALT SCHULLER
// CALLS.C
//
// Walter Schuller
// University of North Florida
//
// 1997


#include <stdlib.h>
#include <string.h>
#include "\Powertcp\include\common.h"
#include <time.h>


//**************************************************************
// delay routine with no cpu sleep


void  delay(long secs)
            {
            long start, finish, ltime;

            start = time(&ltime);
            finish = start + secs;
            while (time (&ltime) < finish);
            return;
            }


//****************************************************************
//  delay routine using cpu sleep
//  USAGE => sleep( (clock_t)3 * CLOCKS_PER_SEC );


void  sleep( clock_t wait )
            {
            clock_t goal;
            goal = wait + clock();
            while( goal > clock() );
            }


//*************************************************************
// Get and return the specified string resource.


LPSTR GetString(HINSTANCE hinst, WORD id)
   {
   static szBuffer[256];
   LoadString(hinst, id, (LPSTR) szBuffer, 255);
   return (LPSTR) szBuffer;
   }



//Dart Communications
//**************************************************************
```

```
BOOL IsEqual(HINSTANCE hinst, int id, LPSTR lpsz)
   {
   return _fstrcmp(GetString(hinst, id), lpsz) == 0;
   }


//Dart Communications
//*************************************************************


LONG GetStringObject(HWND hwnd, int id, LPSTR lpsz, int nSize)
   {
   GetDlgItemText(hwnd, id, lpsz + 2, nSize);
   *((WORD FAR*) lpsz) = _fstrlen(lpsz + 2);
   return (LONG) lpsz;
   }


//Dart Communications
//***********************************************************
// Routine to take a given string and set up for SNMP String Structure
Definition

LONG GetStringProvided(LPSTR lpsz)
   {
    *((WORD FAR*) lpsz) = _fstrlen(lpsz + 2);
   return (LONG) lpsz;
   }


//*************************************************************


LONG GetLongObject(HWND hwnd, int id)
   {
   char szBuffer[11];

   GetDlgItemText(hwnd, id, szBuffer, sizeof(szBuffer));
   return atol(szBuffer);
   }

//Dart Communications
//***************************************************************
//    END OF FILE                          WALT SCHULLER
```

APPENDIX K

Glossary of Terms Used

Acronym	Meaning
ANSI	American National Standards Institute
API	Applications Program Interface
ANS.1	Abstract Syntax Notation 1
BER	Basic Encoding Rules
CMIP	Common Management Information Protocol
COM x	Serial Port x
CTS	Clear-To-Send
DCD	Data-Carrier-Detect
DLL	Dynamic Link Library
DNS	Domain Name Server
DTR	Data Terminal Ready
FTP	File Transfer Program
GPI	General Purpose Interface
HTML	HyperText Markup Language
I/O	Input/Output
ISO	International Organization for Standardization
ISODE	ISO Development Environment
LAN	Local Area Networks
MFC	Microsoft Foundation Class
MIB	Management Information Base
OOP	Object Oriented Programming

RFC	Request For Comments
RTS	Request-To-Send
SDK	Software Development Kit
SMIC	SNMP Management Information Compiler
SMICng	SNMP Management Information Compiler - Next Generation
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TELNET	Telecommunications Network
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol

VITA

Walter Schuller has a Bachelor of Science Degree in Electrical Engineering from the University of North Florida, and expects to receive his Master of Science in Computer and Information Sciences from the same university in December 1997. He also has an Associate in Electronic Technology from Massey Technical College.

At the time of this writing, Walt was employed as an engineering supervisor at W.J.X.T. Post Newsweek. Prior to that, Walt was employed by the Burroughs Corporation, in Jacksonville, as a field service representative and by Vitro Services, located at Eglin Air Base, as an electronic technician working on defense R&D.

Walt has interests in the field of digital communications and networking. He is also quite interested in hardware development and robotics.