PacketView™ User's Manual

Legal Notice

Trademarks

Table of Contents

Introduction

Product Description

PacketView™ is a software product that allows you to view network traffic, debug network drivers and protocols, and learn more about your networking environment. Coupled with a network controller and a packet driver or ODI driver, PacketView™ turns any DOS-based system into a real time protool and network analysis tool. PacketView™ can also be used to view and analyze previously saved packets without a network controller or driver. A unique feature of PacketView™ is that it allows users to independently develop their own protocol decoders (using assembler or a Microsoft C compiler) for use with PacketView™.

System Requirements

PacketView™ requires an IBM or compatible PC/XT/AT or PS/2 system running DOS version 5.00 or above. A hard disk and 640K base memory is recommended, but not required. For real time access to a network, the system must contain a network controller with either a packet driver or ODI driver that supports promiscuous mode.

System Limitations

PacketView™ uses base and expanded (EMS) memory to collect packets from the network. As much base memory as possible should be available when PacketView™ is loaded.

Filters	10 Display Filters and 10 Capture Filters. Each filter consists of up to 5 match terms.
Packets	Defaults to 4096 packets, user may select up 65535, also limited by available memory.
Protocol Decoders	Limited by available memory.
Symbols	Limited by available memory (requires 24 bytes per node symbol or host symbol - OIDs require more).

Customer Support

Technical support for PacketView™ is available by calling Klos Technologies, Inc. support at (607) 753-0568 between 8:00 AM and 5:00 PM EST or via e-mail at support@klos.com.

Installation

Installing PacketView™

To install PacketView™, put the PacketView™ diskette into the A: drive and type "A:INSTALL" at the DOS command prompt. Install will prompt for owner name, company name and the directory in which to copy PacketView™ (default C:\PV). Install then creates the specified directory, if necessary, and copies the PacketView™ files to that directory. If the PacketView™ directory in not in the system PATH, be sure to update the PATH to include the PacketView™ (C:\PV) directory.

Loading the Packet Driver

Most packet drivers are TSR (Terminate and Stay Resident) programs or device drivers with unique command line definitions. Two enhanced packet drivers developed by Klos Technologies, Inc. are included with PacketView™. One for NE1000, NE2000 and compatible Ethernet adapters, and one for COM20020 based ARCNET adapters. These packet drivers are described in Appendix C.

Also included is the CRYNWR Packet Driver Collection. This collection is provided at no cost, and is NOT a part of PacketView™. Appendix D includes support and installation information, as well as examples for loading packet drivers of several common network controller boards. If your controller is not supported by the CRYNWR Packet Driver Collection, refer to the documentation specific to the packet driver for your network controller or contact customer support.

Loading an ODI Driver

If you would like to use an PacketView™ with an ODI driver, it is recommended that you NOT load any networking software except LSL.COM and the actual ODI driver (also known as an MLID). This means that you would NOT load IPXODI (and NETx) or TCPIP or any other protocol stack that would use the same ODI driver you wish to use with PacketView™. Also, try to determine if the ODI driver supports promiscuous mode. This is necessary for proper operation of PacketView™. To load the LSL and ODI driver, follow the instructions in the Novell (or manufacturer's) manual for the specified network adapter. An example for an NE2000 board would look like this:

```
lsl
ne2000
```

Using PacketView™

To run PacketView™, type "PV" at the DOS command prompt. If the PacketView™ program was not copied to a directory in your system's PATH, be sure to change your current directory to the directory in which PacketView™ was copied.

Example:

```
C:\>PV↵
```

When PacketView™ initializes, it determines the location of the PacketView™ program file (PV.EXE) on the hard disk, and assumes all other support files are in the same subdirectory. For example, the default configuration file for PacketView™ is PV.CFG. If PV.EXE is located in the subdirectory C:\PV, PacketView™ will look for PV.CFG in the same directory. The default location for the support files may be overridden by the use of an environment variable called "PACKETVIEW". For example, if PV.EXE is found in the C:\BIN subdirectory, but the support files are in the C:\PV subdirectory, the following line should be included in AUTOEXEC.BAT:

```
SET PACKETVIEW=C:\PV
```

Files that are always kept in the PacketView™ (ex. C:\PV) subdirectory include the configuration files, filter files, and the NODES, HOSTS, VENDORS and OIDS files. However, PacketView™ will look in the current directory for the NODES, HOSTS, VENDORS and OIDS files before it searches the PacketView™ (ex. C:\PV) subdirectory.

Command line options

PacketView™ supports several command line options that allow you to control the initial state of PacketView™ when started.

PV [MONO] [PACKETS=nnnnn] [NOEMS] [NONE | PD=nn | BOARD=nn] [BATCH=filename]

MONO Selects the display characteristics for a monochrome display. Default display mode is color with color display adapters. This is especially useful for LCD screens where color is difficult to see.

PACKETS=nnnnn Sets the maximum number of packets (nnnnn) that can be held in memory at any one time. The maximum number of packets may be from 128 to 65535. The default is PACKETS=4096.

NOEMS Disables the use of Expanded Memory (EMS). The default is to use Expanded Memory when available.

NONE Disables the search for a packet driver stub. This is useful when doing post analysis on saved packet files. The default is to search for a packet driver stub.

[PD=]nn Specifies the specific interrupt for the packet driver stub to be used. nn must be specified in hexadecimal and must be in the range of 60 through 80 inclusive.

BOARD=nn Specifies the ODI driver to be used. nn must be specified in decimal and must be in the range of 1 through 8 inclusive.

BATCH=filename Enables batch mode operation. In this mode, PacketView™ will collect packets until it's memory or packet table is full. It then writes the packets to the filename specified and terminates. Note that if any key is pressed while PacketView™ is executing in batch mode, batch mode is automatically disabled and WILL NOT terminate upon a buffer full condition.

The Main Display

The Main Display provides the primary interface for PacketView™. The display provides current packet information in one of two modes: *Line Mode* or *Detail Mode*. Line mode provides a screen of single line descriptions of packets, while Detail mode provides a complete description of a single packet.

If function key display is enabled, the function key definitions for the Main Display are displayed along the bottom portion of the screen.

Provided on all displays is the number of packets currently in the packet buffer, the percentage of memory used and the packet driver receiver state.

In addition to the two primary modes of display PacketView™ also provides two independently controlled data display modes, RAW and TEXT modes.

RAW Mode

RAW mode displays the data portion of the packet in hexadecimal and ASCII representation. If RAW mode is not in effect, PacketView™ will decode the packet based upon the protocol specified and the decoders available. If

PacketView™ cannot recognize the protocol specified, the packet will be displayed in the RAW mode format.

TEXT Mode

If the contents of the data portion of the packet is all ASCII displayable characters then in TEXT mode this data will displayed in ASCII. If TEXT mode is disabled then the data portion of the packet will be displayed in hexadecimal.

Line Mode

In Line mode, PacketView™ displays a screen of single line descriptions of packets the packet buffer. Only those packets that satisfy any Display Filters will be displayed. If no Display Filters are defined, then all packets are displayed.

```
PacketView v1.23                               Total packets:   332  Memory used:   1%
Copyright, Klos Technologies, Inc.             Receiver state: Enabled

   1)    0.114 0035 DIX: 00000C004493 <- 0800200894E1 [0800] IP: 130.204.5.68 ->
   2)    0.153 002E DIX: 0800200894E1 <- 00000C004493 [0800] IP: 137.39.1.6 -> 1
   4)    0.288 0044 DIX: 0207010DF931 <- 02608C542501 [0BAD] VINES IP:
   5)    0.292 003A DIX: 02608C542501 <- 0207010DF931 [0BAD] VINES IP:
   7)    0.369 004E DIX: 0207010C11ED <- 02608C542501 [0BAD] VINES IP:
   8)    0.373 002E DIX: Broadcast    <- 080089A17562 [809B] AppleTalk:
   9)    0.378 0031 802.3: 090007FFFFFF <- 080089A17562 [0031] DSAP=AA SSAP=AA C
  10)    0.592 0060 802.3: Broadcast    <- DEMO         [0060] IPX: SAP:
  11)    0.613 002E DIX: 0207010DF931 <- 02608C542501 [0BAD] VINES IP:
  13)    0.652 002E 802.3: 020701058A6A <- BLUE         [0020] IPX: Unknown:
  14)    0.657 003C DIX: Broadcast    <- 08001E016136 [0800] IP: 130.204.8.9 ->
  15)    0.827 0060 802.3: 0307011C1C1C <- 020701074C2F [0060] DATA: 80 80 00 00
  16)    0.887 0035 DIX: 0800200894E1 <- AA000400FFFF [0800] IP: 130.45.4.100 ->
  17)    0.908 0032 DIX: Broadcast    <- 02070101E458 [0806] ARP: (0800) REQUEST
  19)    0.986 0057 DIX: AA000400FFFF <- 0800200894E1 [0800] IP: 130.204.5.68 ->
  21)    1.034 004E DIX: 0207010C11ED <- 02608C542501 [0BAD] VINES IP:
  22)    1.114 0030 DIX: 0800200894E1 <- AA000400FFFF [0800] IP: 130.45.4.100 ->
  23)    1.122 0228 DIX: AA000400FFFF <- 0800200894E1 [0800] IP: 130.204.5.68 ->
+--F1------F2------F3------F4------F5------F6------F7------F8------F9------F10-+
¦ HELP   ¦ EDIT   ¦PACKET ¦RESTART¦TOGGLE ¦       ¦ PRINT ¦ GOTO  ¦CONTIN-¦ MAIN ¦
¦        ¦FILTERS¦REPLAY ¦RECEIVE¦RECEIVE¦       ¦       ¦PACKET ¦ UOUS  ¦ MENU ¦
+-----------------------------------------------------------------------------+
```

Typical Line Mode Display

The first column contains the packet number. Each packet in the packet buffer is assigned a sequential number for reference.

The second column indicates the time for the packet in one of three formats. It may indicate either a capture time (the time the packet was received),a relative time (time before or after an event marker) or a delta time (time between adjacent packets).

The third column contains the size of the packet in hexadecimal. This size indicates the size of the *data* field for the packet, and does not include the MAC header or CRC bytes.

The remaining columns can be optionally removed from the display, or vary in format based upon the media and protocols involved.

Optionally, the MAC (Media Access Control) layer information may be displayed. For Ethernet, this includes the media descriptor (DIX or 802.3), the destination address, the source address and the type field. For token-ring, the display indicates the media descriptor (802.5), the destination address and the source address. For ARCNET, the display indicates the media descriptor (ARC), the destination node ID, the source node ID, and the protocol ID.ND -> UD

The format of the remainder of the line will vary based upon the protocol indicated and whether or not raw mode or text mode are in effect.

Detail Mode

Detail mode, PacketView™ provides a complete description of a single packet. The display includes the MAC layer information, the packet number, packet size and the time for the packet in one of three formats. The time field may indicate either a capture time (the time the packet was received),a relative time (time before or after an event marker) or a delta time (time between adjacent packets).

```
PacketView v1.23                          Total packets:   611  Memory used:  12%
Copyright, Klos Technologies, Inc.        Receiver state: Disabled

    IEEE 802.3:
        Destination: 0207010E8A4E  Size: 002E  Number:    20
        Source:      0207010516F6  Type: 0028  Time:   25.65

    IPX:
    Checksum = FFFF, Packet length = 0028
    Transport control: 0  Hop Count: 0
    Protocol type is 1 (RIP)
    Destination address: 00000001.0207010E8A4E
    Destination socket:  0453 (Routing Information Protocol)
    Source address:      00000001.0207010516F6
    Source socket:       0453 (Routing Information Protocol)

    RIP:  Route Response
          Network   Hops  Ticks
          --------  ----  -----
          00000002    1      2

+--F1------F2------F3------F4------F5------F6------F7------F8------F9------F10-+
¦ HELP  ¦ EDIT  ¦PACKET ¦RESTART¦TOGGLE ¦       ¦ PRINT ¦ GOTO  ¦CONTIN-¦ MAIN ¦
¦       ¦FILTERS¦REPLAY ¦RECEIVE¦RECEIVE¦       ¦       ¦PACKET ¦ UOUS  ¦ MENU ¦
+------------------------------------------------------------------------------+
```

Typical Detail Mode Display

Key Functions

The following keys are defined for both Line and Detail Mode displays:

Key	Action
Alt-X	Exit PacketView™
Esc	Exit a menu or function
Return	Toggle between Detail Mode and Line Mode
C	Toggle color mode between COLOR and MONOCHROME
D	Set the display to Detail Mode
E	Toggle Error Mode, when enabled packets with errors are dropped, when disabled all packets are saved
F	Toggle the display of the Function Key definitions on the bottom of the screen
L	Set the display to Line Mode
M	Toggle display of MAC information
R	Toggle between raw data display and protocol decode display
Ctrl-R	Switch to "super"-RAW mode, displaying all bytes of the actual frame in hexadecimal and ASCII
S	Toggle the node name, host name and OID between symbolic definitions and numeric values

Alt-S	Send the current packet
T	Toggle the packet time field between absolute time and delta time since the previous packet
Ctrl-T	Mark an event. All other packets' time field will be relative to this packets' time field
Alt-T	Toggle Text Mode, wherever packet data would normally display the data in hexadecimal, the data is checked for displayable text, if all of the data is displayable then it is displayed in ASCII
U	Start Update mode, keep the display current with the last packet received
V	Toggle the display mode between 25 and 50 lines for VGA displays

Function Key	Action
F1	Display help
F2	Enter the Filter Menu
F3	Enter the Replay Menu
F4	Clear the packet buffer and enable the receiver to capture packets from the network
Alt-F4	Start "continuous capture" mode. This causes the packet buffer to be cleared and the receiver to be enabled. Whenever the packet buffer becomes full, it will automatically be cleared again, restarting the capture. This mode is terminated whenever any key is pressed.
F5	Toggle enable/disable of packet capture from the network
F6	(undefined at this time)
F7	Select and print a range of packets
F8	Go to a packet by number
F9	Enable Continuous Capture
F10	Display the Main Menu

The following keys are defined for the Line Mode display:

Key	Line Mode Action

Home	Move the cursor to the first packet on the screen, if the cursor is already on the first packet then the cursor is moved to the first packet in the buffer
End	Move the cursor to the last packet on the screen, if the cursor is already on the last packet then the cursor is moved to the last packet in the buffer
PgUp	Move the cursor up one screen load of packets
PgDn	Move the cursor down one screen load of packets
North	Move the cursor up one packet
South	Move the cursor down one packet

The following keys are defined for the Detail Mode display:

Key	Detail Mode Action
Home	Move the cursor to the first packet
End	Move the cursor to the last packet
PgUp	Display the previous packet
PgDn	Display the next packet
North	Move the cursor up one line in the current packet display
South	Move the cursor down one line in the current packet display
Ctrl-Home	Display the first screen of the current packet
Ctrl-End	Display the last screen of the current packet
Ctrl-PgUp	Display the previous screen for the current packet
Ctrl-PgDn	Display the next screen for the current packet

Main Menu

The PacketView™ Main Menu is selected from the Main Display by pressing the F10 key. This screen provides the ability to load and save PacketView™ configuration information, to load and save the contents of the current packet buffer.

It also provides basic system resource information. This information includes the base and expanded memory available for storing symbols and packets, the packet driver interrupt, the maximum number of packets that may be held in memory at once, the network physical layer type, and the number of node and host symbols currently defined.

```
PacketView 1.23                          Total packets:     0  Memory used:   0%
Copyright, Klos Technologies, Inc.       Receiver state: Enabled


                                  Main Menu

                       F1  - Help
                       F2  - Load Configuration from Disk
                       F3  - Save Configuration to Disk
                       F4  - Load Packet Buffer from Disk
                       F5  - Save Packet Buffer to Disk
                       F10 - Display Protocol List





                      Serial number: 01000001
                      Registered to: Patrick Klos
                                     Klos Technologies, Inc.




    Available base memory: 323K            Packet driver at interrupt 0x60+
    Available expanded memory: 2048K       Packet list contains 16384 entries
    0 node symbols using 0 bytes           Current media type is Ethernet
    0 host symbols using 0 bytes
```

Main Menu

The following describes each of the functions:

F1 - Help

Provides the current help information for the Main Menu.

F2 - Load Configuration from Disk

This function loads predefined configuration information for PacketView™. This configuration information includes the following configuration options:

Color/Mono	Controls the use of color for the display.
Function Lines	Controls the display of the function key definitions on the Main Display.
Time Display Mode	Controls the packet time display format.
Display Mode	Selects either the line or detail modes for the Main Display.
Screen Mode	Selects the number of lines displayed for the Main Display on EGA/VGA systems.

Symbolic Mode	Selects whether symbols are to be displayed.
Packets	Maximum number of packets that may be held in memory at any one time.

The new configuration information takes effect immediately **except for the Packets value** which takes effect only upon initialization of PacketView™.

F3 - Save Configuration to Disk

This function saves the current configuration information for PacketView™. See "Load Configuration" for a description of the configuration options to be saved. This configuration can then be loaded again at a later time.

F4 - Load Packet Buffer from Disk

Clears the packet buffer, then loads packets into the packet buffer from the file specified. Any currently enabled Capture Filter(s) will be applied to the packets from the disk and only those passing the Capture Filter(s) will be placed into the packet buffer.

F5 - Save Packet Buffer to Disk

Saves the packet buffer contents to the file specified. Any currently enabled Display Filter(s) will be applied to the packets from the packet buffer and only those passing the Display Filter(s) will be placed into the output file. The default extension of .PVD is added when no extension is given.

F10 - Display Protocol List

This option will display a list of the protocol decoders loaded along with some memory usage information. This list will include any custom protocol decoders that have been loaded.

Edit Filter Menu

PacketView™ provides the mechanism, using Filters, to selectively start and stop packet collection, store and view packets received from the network or selectively read packets from a packet file. The "Trigger Filter" is used to start or stop packet collection. When a packet is received that matches the filter criteria then packet collection is either started, stopped or toggled according to the filter definition. Trigger filters are used to watch for a specific event on the network then to use either start or stop packet collection thus reducing the packets collected to those just around the significant event. The "Capture Filter" selects which network packets received from the network driver or read from a packet file will be kept in the packet buffer. Packets from the network that are rejected by the capture filter are dropped and can not be retrieved later. Capture filters are useful when it is necessary to collect only specific types of packets from the network. The "Display Filter" selects which packets from the packet buffer will be displayed (or saved when the packet buffer is saved to a file). Since the packets remain in the packet buffer once captured, it is possible to modify display filters without losing packets from the packet buffer. Display filters are used to view specific packet types from the packet buffer without losing packets not of immediate interest.

```
PacketView 1.23                           Total packets:     0  Memory used:   0%
Copyright, Klos Technologies, Inc.        Receiver state: Enabled



                                 Edit Filter Menu

                            F1 - Help
                            F2 - Edit Trigger Filters
                            F3 - Edit Capture Filters
                            F4 - Edit Display Filters
```

Edit Filter Menu

Trigger Filter, Capture Filter and Display Filter Screens

The Filter Screens provide the ability to define, modify, remove, load and save up to ten (10) Trigger, ten (10) Capture and ten (10) Display filters. Each filter allows for specific fields of the packet to be checked and either used to start or stop packet collection (Trigger Filters), saved in the packet buffer (Capture Filters) or displayed on the screen (Display Filters).

Note: The packet receiver is disabled while editing Capture Filters, but remains enabled while editing Display and Trigger Filters.

```
PacketView v1.23                          Total packets:     0  Memory used:   0%
Copyright, Klos Technologies, Inc.        Receiver state: Disabled

                                 Capture Filters

    Filter number 0: (enabled)
        Data at offset 0 is 45XXXXXXXXXXXXXX
        Data at offset 14 is 00AXXXXXXXXXXXXX

    Filter number 1: (enabled)
        Data at offset 0 is 45XXXXXXXXXXXXXX
        Data at offset 16 is 00AXXXXXXXXXXXXX








+--F1------F2------F3------F4------F5------F6------F7------F8------F9------F10-+
¦ HELP  ¦  ADD  ¦ EDIT  ¦DELETE ¦ENABLE ¦DISABLE¦       ¦ LOAD  ¦ SAVE  ¦ DONE ¦
¦       ¦       ¦       ¦       ¦       ¦       ¦       ¦       ¦       ¦      ¦
+------------------------------------------------------------------------------+
```

Filter Screen

The following describes each of the functions:

F1 - Help

Provides the current help information for the Filter Screens.

F2 - Add

The Add function defines a new filter, either Capture or Display. See Filter Editing below.

F3 - Edit

The Edit function allows previously defined filters to be modified. After the filter to be modified has been selected, the functions available to modify the filter are the same as those defined for the Add function. See Filter Editing below.

F4 - Delete

The Delete function allows you to delete a currently defined filter.

F5 - Enable

The Enable function allows you to selectively enable a currently defined filter. Any number of filters may be enabled or disabled at any given time.

F6 - Disable

The Disable function allows you to selectively disable a currently defined filter. This is useful when you want to disable the actions of a specific filter without deleting the filter. The filter may be re-enabled again with the Enable function (see above) at some time later. Any number of filters may be enabled or disabled at any given time.

F8 - Load Filters

Loads previously defined filters from disk. If filters are currently defined, a prompt is provided to ask if the current filters should be kept. If "no" is selected at the prompt then the current filters are deleted before loading the filters from the file. If yes is selected, the filters from disk are added to the list of currently active filters. If the total number of filters exceeds ten (10) then only the first ten (10) filters will be kept.

F9 - Save Filters

This function saves the current set of filters to a file.

F10 - Done

Returns to the Main Menu.

Filter Editing

A filter is a list of up to five (5) match criteria which must all be true for a packet to be selected by the filter. Packets must be selected by at least one filter to be saved or displayed.

Filter Editing provides the ability to set or modify the each of the selection criteria for either a capture filter or a display filter. This is done with the following selection criteria functions:

F1 - Help

Provides the current help information for the Filter Editing.

F2 - Add

The Add function allows a packet match criteria to be added to the filter. However, a maximum of five (5) packet match criteria may be used in any filter.

F3 - Edit

The Edit function allows the current packet match criteria to be changed. The criteria to be edited must be selected using the cursor up and down arrows before selecting the Edit function. The current packet match criteria is always displayed in reverse video.

F4 - Delete

This function deletes the current packet match criteria from the filter. The criteria to be deleted must be selected using the cursor up and down arrows before selecting the Delete function. The current packet match criteria is always displayed in reverse video.

F5 - Negate

This function negates the current packet match criteria. Any packets that the criteria would have rejected are now accepted. Packets that would have been accepted are now ignored. The criteria to be negated must be selected using the cursor up and down arrows before selecting the Negate function. The current packet selection criteria is always displayed in reverse video.

F7 - Lookup Node

This function provides for the lookup of a defined Node name in the symbol table. If the Node is found, its address may be retrieved for use in the filter definition.

F8 - Lookup Host

This function provides for the lookup of a defined Host name in the symbol table. If the Host name is found, its address may be retrieved for use in the filter definition.

F9 - Lookup Vender

This function provides for the lookup of a defined Vender name in the symbol table. If the Vender is found, its information may be retrieved for use in the filter definition.

F10 - Save

After a filter has been completely specified, the Save function saves it for use.

Match Criteria

Each filter may have from one (1) to five (5) match criteria. These criteria may be, "Data" match or "Packet Type" match. Match criteria may be used more than once in a single filter as long as the total number of match criteria for a filter does not exceed five (5). An example might use two Data match criteria in a single filter.

F1 - Data Match

This match criteria prompts for an offset within the data field (4 digit hexadecimal value) of the packet at which to begin matching data. The data field is defined to start after all standard datalink headers, including 802.2 and SNAP headers. Once the data offset has been entered up to 16 hexadecimal digits (8 bytes) of data to be matched is entered. An 'X' in any digit of the match data matches all possible values for the digit.

F2 - Packet Type

The Packet Type criteria compares the 4 digit hexadecimal packet type provided and selects the packet if the packet type matches.

Symbol Lookup During Filter Definition

When entering match criteria data, a symbol lookup will allow the user to insert the value of a symbol into the match criteria data. This is accomplished by pressing either function key F6, F7 or F8, depending on the type of symbol being used. Use F6 for node addresses from the NODES file, F7 for a host address from the HOSTS file, and F8 for a vendor ID from the VENDORS file. To insert a value from a symbol, press the appropriate function key, select the desired symbol using the cursor keys, PgUp or PgDn, then press ENTER. The value for the specified symbol will be entered into the match criteria.

Packet Replay Menu

```
PacketView v1.23                          Total packets:    79  Memory used:   0%
Copyright, Klos Technologies, Inc.        Receiver state: Enabled

Current Packet:    79



                               Packet Replay Menu

                          F1 - Help

                          F2 - Replay Current Packet (Alt-F2 to View)
                          F3 - Replay Packet Range
                          F4 - Change Replay Loop Count
                          F5 - Change Packet Gap
                          F6 - Change Packet Range




Replay Loops: 1
Packet Gap:   [ Actual ]
Packet Range: [ 1 - 79 ]
```

Packet Replay Menu

F1 - Help

F2 - Replay Current Packet (Alt-F2 to View)

F3 - Replay Packet Range

F4 - Change Replay Loop Count

F5 - Change Packet Gap

F6 - Change Packet Range

Files

PV.CFG

The default configuration file for PacketView™. This file is loaded by default whenever PacketView™ is loaded. To modify your default configuration, use the "Save Configuration to Disk" function of the Main Menu after you have selected your preferred configuration. The configuration options maintained by this file include the color mode, function key display, time display format, screen mode (25 or 50 lines), symbolic display mode, and maximum packet count.

HOSTS

The **HOSTS*** file is a text file that provides symbolic name definitions for TCP/IP hosts. The format of each line of the file is a follows:

###.###.###.###	Host_Name

where

###	Decimal value from 0 to 255 (decimal)
Host_Name	Arbitrary name for the host machine. Up to 15 characters, without spaces.

Example:

A host whose name is "ftp.klos.com" and whose IP address is 192.80.49.2 would be entered in the **HOSTS** file as follows:

```
#
#  Entry for ftp.klos.com
#
192.80.49.2    ftp.klos.com
```

Blank lines and lines beginning with the '#' character are ignored as comment lines.

NODES

The **NODES** file is a text file that provides symbolic name definitions for 12 digit hexadecimal (48-bit) network node addresses. The format of each line of the file is as follows:

############	Node Name

* The HOSTS file is similar in format to the standard TCP/IP HOSTS file.

where

############	Refers to a twelve digit hexadecimal number (Each digit between 0 - 9 or A - F.)
Node Name	The name assigned to the node (12 characters, spaces allowed).

Example:

It is usually useful to assign names to file servers and workstations. A file server whose name is "FS1" would be quickly recognized if it's node address had a name associated with it. If the file servers node address is 0207010EF0F4, the **NODES** file would contain the following:

```
#
#  Entry for FS1
#
0207010EF0F4       FS1
```

Blank lines and lines beginning with the '#' character are ignored as comment lines.

VENDORS

The **VENDORS** file is a text file that provides symbolic translations for the 24-bit vendor specific portion of 48-bit node address. The format of each line of the file is as follows:

Vendor_Name

where

######	Six digit hexadecimal number corresponding to the assigned vendor ID for the specified vendor.
Vendor_Name	A six character representation for the specific vendor.

Example:

3Com's vendor ID (assigned by IEEE) is 02608C (hex). To specify this in the VENDORS file, insert the following line:

```
#
#  Entry for 3Com
#
02608C      3Com
```

Blank lines and lines beginning with the '#' character are ignored as comment lines.

OIDS

The **OIDS** file is a text file that provides symbolic definitions for SNMP object IDs. This makes viewing SNMP packets much easier. The format of each line of the file is as follows:

OID_Name ##.##.##.##.##

where

OID_Name	A symbolic name to be used in place of the object ID prefix.
##.##.##.##.##	A object ID prefix in dotted decimal notation. The object ID may have up to 128 32-bit values.

Example:

Here are a few standard SNMP object IDs:

```
iso         1
org         1.3
dod         1.3.6
internet    1.3.6.1
mgmt        1.3.6.1.2
mib-2       1.3.6.1.2.1
```

Blank lines and lines beginning with the '#' character are ignored as comment lines.

Customizing PacketView™

External Protocol Decoders

PacketView™ supports custom external protocol decoders. These external protocol decoders can be developed using most C compilers. Source code for a sample external protocol decoder is provided on the PacketView™ diskette, as well as in Appendix B of this manual.

All external protocol decoders must be written in LARGE model, assuming DS does NOT equal SS, and the decoder's entry points must be forced to load DS upon entry. The MAKEMSC and MAKEBC files show the proper options to use with the Microsoft and Borland C compilers respectively.

These are four main components in an external protocol decoder. These include the *protocol structure*, the *initialization* routine, the *format line* routine, and the *format detail* routine.

The *protocol structure*

The protocol structure (see "structs.h" in Appendix B) provide the interface between the external protocol decoder and PacketView™. It includes the name of the protocol being decoded, the type values that identify the protocol for various frame types, the address of the routine to be called when displaying the protocol in line mode, and the address of the routine to be called when displaying the protocol in detail mode.

The *initialization* routine

The *initialization* routine's primary function is to return the address of the protocol decoder's protocol structure to PacketView™. The initialization routine can link several protocol structures together forming a list of protocols to be handled by the decoder. This is necessary for those protocol decoders that will support more than one protocol. The initialization routine can also load any necessary data (i.e.

tables) from disk using the **open**, **read**, **lseek**, and **close** routines. The initialization routine should have a C function definition as follows:

```
struct protocol * _loadds init()
   {
   /*  body of init() routine  */
   }
```

Note that the name used **must** be "init", as that is what the header file (HEADER.ASM) will be calling to initialize the decoder.

The *format line* routine

The *format line* routine is called whenever a protocol is to be displayed in line mode. In this mode, each packet is summarized in a single line on the screen, allowing information about many packets to be displayed on a single screen. The format line routine is passed three parameters: the address of a character buffer into which the null-terminated single-line description is to be placed, the address of the packet buffer containing the packet contents, and the length of the packet buffer in bytes. In general, the format line routine will use a special form of *sprintf()* to fill the line buffer with the desired information to describe the packet. The format line routine should have a C function definition as follows:

```
void _loadds format_xyz_line(line, packet,
   length)
   char *line;
   byte *packet;
   int length;
   {
   /* body of format_xyz_line() routine */
   }
```

The actual name used for the format line routine is arbitrary since it is only referenced through the protocol structure.

The *format detail* routine

The *format detail* routine is called whenever a protocol is to be displayed in detail mode. In this mode, each packet is displayed with as much information as possible (or necessary) to describe the packet. The format detail routine is passed two parameters: the address of the packet buffer containing the packet contents, and the length of the packet buffer in bytes. In general, the format detail routine will use a special form of *printf()* to present the packet to the user. Technically, the printf routine provided will be formatting the data as requested by the decoder, and putting that data into an internal "screen buffer", which is then manipulated by PacketView™ to allow the user to scroll the packet through the available lines on the screen. This mechanism also allows for printing of packets in the same form as they are displayed on the screen. The format detail routine should have a C function definition as follows:

```
void _loadds format_xyz(packet, length)
   byte *packet;
   int length;
   {
   /*  body of format_xyz() routine  */
   }
```

The actual name used for the format detail routine is arbitrary since it is only referenced through the protocol structure.

Library routines for external protocol decoders

The following routines are provided by PacketView™ to aid in the formatting of packet information:

```
sprintf
printf
format_protocol
format_protocol_line
format_raw
format_raw_line
set_color
falloc
open
read
write
lseek
close
```

The following variables are provided by PacketView™ to aid in the formatting of packet information:

```
home_dir
current_level
```

sprintf

```
char *sprintf(buffer, format, ...)
   char *buffer;
   char *format;
```

The sprintf routine uses the `format` string to format the text and variables specified into the character `buffer`. This routine works very similar to the standard C sprintf routine with a few exceptions. See the section on *printf()* for a description of the available format characters. This routine returns the address of the end of the buffer. This is a quick way to advance the pointer to the end of the buffer when you may want to append more information to the line buffer.

printf

```
void printf(format, ...)
   char *format;
```

The printf routine uses the `format` string to format the text and variables specified into the internal screen buffer. This routine works very similar to the standard C printf routine with a few exceptions. The format characters supported in the PacketView™ version of printf and sprintf are defined as follows:

Control letter(s)	Description of function
%	Display '%' character
b	Format an unsigned binary integer
lb	Format an unsigned long binary integer
d	Format signed decimal integer
ld	Format long signed decimal integer
D	Format long signed decimal integer
u	Format unsigned decimal integer
lu	Format long unsigned decimal integer
x	Format hexadecimal integer
lx	Format long hexadecimal integer
X	Format long hexadecimal integer

m	Format a hexadecimal byte with a mask, the first value is the hexadecimal byte and the second value is the mask. If the corresponding nibble of the mask is 0 then 'X' is output, otherwise the hexadecimal nibble is displayed.
s	Format string
c	Format character
t	Format the long tick/time value to a fixed point decimal value

The following formats provide for a standard display of network values and for symbolic substitution when the value matches a defined symbol.

Control letter(s)	Description of function
i	Format IP address as a dotted decimal number or replace with the symbolic name
n	Format node address as a 12 digit hexadecimal number or replace with the symbolic name.
o	Format OID as a dotted numeric value or replace with the symbolic name

Formatting IP Addresses

The 'i' format takes a 32-bit unsigned long (dword) parameter and will display the IP address represented by the 32-bit value in the decimal-dotted notation, always padding to a display width of 15 characters. For example, if the parameter for the 'i' format contained the value 0xc0503101, the resulting string will be '192.80.49.1   ' (4 trailing spaces). If symbolic mode is enabled, the IP address will be looked up in the IP address symbol table. If found, the first 15 characters of the symbol representing the IP address will replace the dotted-decimal notation; otherwise the dotted-decimal notation will be used.

Formatting 48-bit Node Addresses

The 'n' format takes a byte pointer as a parameter and will display the node address represented by the 48-bit (6 byte) value pointed to by the byte pointer in hexadecimal format. If symbolic mode is enabled, the node address will be looked up in the node address symbol table. If found, the first 12 characters of the symbol representing the node address will replace the hexadecimal format. If not found, the high 24-bits of the node address are looked up in the vendor address symbol table. If the vendor portion of the address has a corresponding symbolic representation, the first 6 characters of the symbol will replace the first 6 characters of the hexadecimal node address, followed by the remaining 6 hexadecimal digits of the node address. Otherwise, the entire node address will be displayed in hexadecimal format.

Formatting Object IDs

The 'o' format takes a pointer to an OID structure and will display the object id represented in the standard dotted decimal notation. If symbolic mode is enabled, the object id will be looked up in the object id table. If found, the part of the id that is defined will be displayed in place of the dotted decimal notation. If a suffix portion is not found will be displayed in dotted decimal notation.

format_protocol

```
      void format_protocol(packet, length, type,
         media)
         byte *packet;
         int length;
         word type;
         word media;
```

The format_protocol routine allows a protocol decoder to "hand off" a packet (or portion of a packet) to another protocol decoder to be decoded as a different protocol. This is especially useful when supporting protocol tunneling (one protocol is carried within another). The parameters to this routine include the address of the packet buffer, the packet buffer's length, the desired packet type, and the media value for which the packet type is defined.

format_protocol_line

```
void format_protocol_line(buffer, packet,
   length, type, media)
   char *buffer;
   byte *packet;
   int length;
   word type;
   word media;
```

The format_protocol_line routine allows a protocol decoder to "hand off" a packet (or portion of a packet) to another protocol decoder to be decoded as a different protocol. This is especially useful when supporting protocol tunneling (one protocol is carried within another). The parameters to this routine include the address of the line buffer, the address of the packet buffer, the packet buffer's length, the desired packet type, and the media value for which the packet type is defined.

format_raw

```
void format_raw(heading, packet, length)
   char *heading;
   byte *packet;
   int length;
```

The format_raw routine allows a protocol decoder to display a packet (or portion of a packet) as a simple hexadecimal dump of the contents. The parameters to this routine include the address of the text string to display as the header, the address of the packet buffer, and the packet buffer's length. If text mode is enabled, then the data will be examined to see if the entire buffer can be displayed as text, if so it will be displayed as text, otherwise it will be displayed in hexadecimal.

format_raw_line

```
void format_raw_line(buffer, packet,
   length)
```

```
        char *buffer;
        byte *packet;
        int length;
```

The format_raw_line routine allows a protocol decoder to display a packet (or portion of a packet) as a simple hexadecimal dump of the contents. The parameters to this routine include the address of the line buffer, the address of the packet buffer, and the packet buffer's length. If text mode is enabled, then the data will be examined to see if the entire buffer can be displayed as text, if so it will be displayed as text, otherwise it will be displayed in hexadecimal.

set_color

```
    void set_color(background, foreground)
        int background;
        int foreground;
```

The set_color routine allows a protocol decoder (either line or detail mode) to select the background and foreground colors to be used to display the information relating to the current packet. In PacketView™, the color attribute is allocated on a per-line basis. Colors cannot be changed in the middle of a line. In detail mode, separate lines may have different colors. In line mode, the last set_color() call determines the color that will be used to display the line. See the structs.h file for definitions for the various colors.

falloc

```
    byte *falloc(size)
        int size;
```

The falloc routine is used by protocol decoders during initialization time only. It allows a protocol decoder to allocate memory from the PacketView™ memory pool for whatever the protocol decoder may deem necessary. The size parameter specifies the size in bytes of the area to be allocated.

open

```
int open(filename, mode)
   char *filename;
   int mode;
```

The open routine uses the DOS function 0x3d to open the file specified by the filename. The file is opened with the mode specified (0 = read only, 1 = write only, 2 = read/write). If the file open is successful, the file handle is returned; otherwise a -1 is returned.

read

```
int read(handle, buffer, length)
   int handle;
   char *buffer;
   int length;
```

The read routine uses the DOS function 0x3f to read bytes from the file specified by the file handle. The parameters include the file handle (as returned by the open function), the address of the buffer, and the length of the buffer. Note that this function uses the DOS read file function. There is no interpretation of the data (including new-line/carriage-return-line-feed conversions). This function returns -1 if an error occurs, or the number of bytes of data read from the file into the buffer.

write

```
int write(handle, buffer, length)
   int handle;
   char *buffer;
   int length;
```

The write routine uses the DOS function 0x40 to write bytes to the file specified by the file handle. The parameters include the file handle (as returned by the open function), the address of the buffer, and the length of the buffer. Note that this function uses the DOS write file function. There is no interpretation of the data (including new-line/carriage-return-line-feed conversions). This function returns -1 if an error occurs, or the number of bytes of data written into the buffer from the file.

lseek

```
long lseek(handle, offset, where)
   int handle;
   long offset;
   int where;
```

The lseek routine is used to position the file specified by the file handle to a specific location. The parameters include the file handle, the long offset specifying the new position in the file, and the control value indicating where the offset is relative to. The where values are 0 for beginning of file, 1 for current position, and 2 for the end of the file. The lseek routine return -1L if an error occurs, or the long offset of the new current position of the file.

close

```
int close(handle)
   int handle;
```

The close function closes the file specified by the file handle. If an error occurs, -1 is returned.

home_dir

```
char *home_dir;
```

The home_dir variable contains the address of the PacketView™ home directory (the directory the PV.EXE file is located in).

current_level

```
int current_level;
```

The current_level variable contains the current stack level being decoded. It's purpose is not currently defined for external protocol use.

Assistance with external protocol decoders

Technical support for PacketView™ is available by calling Klos Technologies, Inc. support at (607) 753-0568 between 8:00 AM and 5:00 PM EST or via e-mail at support@klos.com. Custom protocol decoders can be developed for a nominal fee, contact technical support.

Glossary

AppleTalk®	A set of protocols defined by Apple Computer.
ARCNET®	A self-polling "modified token passing" network operating at a 2.5M bit data rate.
Blue Book Ethernet	The original Ethernet definition produced by Digital Equipment Corporation, Intel Corporation and Xerox Corporation (DIX). Most notably differing from IEEE 802.3 by defining the type field as the protocol ID rather than the data length.
Capture Filter	Determines which packets from the network or packet file will be stored in the packet buffer.
CSMA/CD	Carrier Sense Multiple Access/Collision Detection - A network physical layer method used to control media access in a bus topology.
DECnet®	A suite of protocols defined by Digital Equipment Corporation.
Display Filter	Determines which packets from the packet buffer will be displayed or saved to disk.
Ethernet	A 10 megabit per second baseband bus topology network originally developed by Xerox Corporation.
EtherTalk®	AppleTalk on Ethernet.
Filter	Provides the means to select and reject packets.
FTP	File Transfer Protocol for TCP/IP.
IPX/SPX	Internetwork Packet eXchange/Sequenced Packet eXchange protocols used by Novell.
ISDN	Integrated Services Data Network - digital communication services provided by telephone companies
LocalTalk®	Low speed AppleTalk for personal computers.

MAC	Media Access Control - A datalink layer protocol controlling access to the physical layer.
MS-NET	A network operating system produced by Microsoft.
NetBIOS	Network Basic Input/Output System - A protocol and system interface for data exchange and network access.
Sun NFS®	Network File System - A network operating system based on TCP/IP and produced by Sun Microsystems.
Novell NetWare®	The file server based network operating system produced by Novell.
Packet Buffer	The memory used to hold packets received from the network or a file.
Packet Driver	A standard software interface to a network controller.
PPP	Point-to-Point Protocol.
Promiscuous Mode	Network controller mode where the network controller passes every packet on the network to the packet driver, regardless of intended destination.
Protocol	A set of rules used to govern how two or more computers communicate on a network.
Protocol Decoder	External software procedure(s) loaded by a Klos protocol analyzer to allow alternative and additional protocol display.
SLIP	Serial-Line IP
SNA	Systems Network Architecture - A suite of protocols defined by IBM for mainframe communications.
StarLAN	A network operating system produced by AT&T.
TCP/IP	Transmission Control Protocol/Internet Protocol
Token-Ring	A network physical layer interface that uses a token message passed around a ring of computers to arbitrate network access.

TSR	Terminate and Stay Resident, a program which remains in memory after it terminates. Typically the program then provides services to other programs via a mutually agreed upon protocol or interface.
VINES®	A network system produced by Banyan Systems.
XNS	Xerox Network Systems - A suite of protocols defined by Xerox Corporation.
X Windows	A workstation windowing system produced by the Massachusetts Institute of Technology part of which includes a network protocol.

Appendix A - Software License Agreement

PacketView™

SOFTWARE LICENSE AGREEMENT

- READ THIS BEFORE USE -

Please read this License carefully.

You are purchasing a license to use the PacketView™ Software. The Software is owned by and remains the property of Klos Technologies, Inc., is protected by international copyrights, and is transferred to the original purchaser and any subsequent owner of the Software media for their use only on the license terms set forth below. Opening the packaging and / or using PacketView™ indicates your acceptance of these terms. If you do not agree to all of the terms and conditions, or if after use you are dissatisfied with your PacketView™ Software, return the Software, manuals and any partial or whole copies within thirty days of purchase to the party from whom you received it for a refund, subject to our restocking fee.

<u>Grant of License.</u> Klos Technologies, Inc. ("KTI"), grants the original purchaser ("Licensee") the limited rights to possess and use the Klos Technologies, Inc. Software and User Manual ("Software"), on the terms and conditions specifically set out in this License.

<u>Term.</u> This License is effective as of the time Licensee receives the Software, and shall continue in effect until Licensee ceases all use of the Software and returns or destroys all copies thereof, or until automatically terminated upon the failure of Licensee to comply with any of the terms of this License.

<u>Your Agreement.</u>

- Licensee is granted a license to use the Software for its intended purposes. Licensee agrees that the Software will be used solely for Licensee's internal purposes, and that at any one time, the Software will be installed on a single computer only. If the Software is installed on a networked system, or on a computer connected to a file server or other system that physically allows shared access to the Software, Licensee agrees to provide technical or procedural methods to prevent use of the Software by more than one user.

- One machine-readable copy of the Software may be made for BACK-UP PURPOSES ONLY, and the copy shall display all proprietary notices, and be labeled externally to show that the back-up copy is the property of KTI, and that its use is subject to this License. Documentation in whole or part may not be copied.

- Use of the Software by any department, agency or other entity of the U.S. Federal Government is limited by the terms of the attached "U.S. Rider for Governmental Entity Users", which is incorporated by reference into this License.

- Licensee may transfer its rights under this License, PROVIDED that the party to whom such rights are transferred agrees to the terms and conditions of this License, and written notice is provided to KTI. Upon such transfer, Licensee must transfer or destroy all copies of the Software.

- Except as expressly provided in this License, Licensee may not use, copy, disseminate, modify, reverse engineer, distribute, sub-license, sell, rent, lease, lend, give or in any other way transfer, by any means or in any medium, including telecommunications, the Software. Licensee will use its best efforts and take all reasonable steps to protect the Software from unauthorized use, copying or dissemination, and will maintain all proprietary notices intact.

LIMITED WARRANTY. KTI warrants the Software media to be free of defects in workmanship for a period of ninety days from purchase. During this period KTI will replace at no cost any such media returned to KTI, postage prepaid. This service is KTI's sole liability under this warranty.

DISCLAIMER. LICENSE FEES FOR THE SOFTWARE DO NOT INCLUDE ANY CONSIDERATION FOR ASSUMPTION OF RISK BY KTI, AND KTI DISCLAIMS ANY AND ALL LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR OPERATION OR INABILITY TO USE THE SOFTWARE, OR ARISING FROM THE NEGLIGENCE OF KTI, OR ITS EMPLOYEES, OFFICERS, DIRECTORS, CONSULTANTS OR DEALERS, EVEN IF ANY OF THESE PARTIES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. FURTHERMORE, LICENSEE INDEMNIFIES AND AGREES TO HOLD KTI HARMLESS FROM SUCH CLAIMS. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY THE LICENSEE. THE WARRANTIES EXPRESSED IN THIS LICENSE ARE THE ONLY WARRANTIES MADE BY KTI, AND ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE.

THIS WARRANTY GIVES YOU SPECIFIED LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF WARRANTIES, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

General. This License is the complete and exclusive statement of the parties' agreement. Should any provision of this License be held to be invalid by any court of competent jurisdiction, that provision will be enforced to the maximum extent permissible, and the remainder of the License shall nonetheless remain in full force and effect. This License shall be controlled by the laws of the State of New Hampshire, and the United States of America.

Rider For U.S. Governmental Entity Users

This is a Rider to the PacketView™ SOFTWARE LICENSE AGREEMENT, ("License"), and shall take precedence over the License where a conflict occurs.

1. The Software was: developed at private expense; no portion was developed with government funds; is a trade secret of KTI and its licensor for all purposes of the Freedom of Information Act; is "commercial computer software" subject to limited utilization as provided in any contract between the vendor and the government entity; and in all respects is proprietary data belonging solely to KTI and its licensor.

2. For units of the DoD, the Software is sold only with "Restricted Rights" as that term is defined in the DoD Supplement to DFAR 252.227-7013 (b)(3)(ii), and use, duplication or disclosure is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Manufacturer: Klos Technologies, Inc., 604 Daniel Webster Highway, Merrimack, NH 03054.

3. If the Software was acquired under a GSA Schedule, the Government has agreed to refrain from changing or removing any insignia or lettering from the Software or Documentation or from producing copies of manuals or disks (except for backup purposes) and: (1) Title to and ownership of the Software and Documentation and any reproductions thereof shall remain with KTI and its licensor; (2) use of the Software shall be limited to the facility for which it is acquired; and (3) if the use of the Software is discontinued at the original installation and the Government wishes to use it at another location, it may do so by giving prior written notice to KTI, specifying the new location site and class of computer.

4. Governmental personnel using the Software, other than under a DoD contract or GSA Schedule, are hereby on notice that use of the Software is subject to restrictions that are the same or similar to those specified above.

Appendix B - Sample External Protocol Decoder Listings

HEADER.ASM

```
        Page    56,132
        Title   HEADER - Header for protocol decoders

;
;       Written by Patrick Klos
;       Copyright, Klos Technologies, Inc.
;

DGROUP  Group   _DATA

PARMS   Struc
        Dw      ?                       ;BP
        Dw      ?                       ;IP
        Dw      ?                       ;CS
PARM1   Dw      ?                       ;
PARM2   Dw      ?                       ;
PARM3   Dw      ?                       ;
PARM4   Dw      ?                       ;
PARMS   Ends

HEADER_TEXT Segment Byte Public 'CODE'
HEADER_TEXT Ends

_DATA   Segment Word Public 'DATA'

        Public  __acrtused
__acrtused Dw   0                       ;

        Public  _decoder_header
_decoder_header Dd HEADER_TEXT:decoder_header

_DATA   Ends

CONST   Segment Word Public 'CONST'
CONST   Ends

_BSS    Segment Word Public 'BSS'
_BSS    Ends

        Extrn   _init:Far

HEADER_TEXT Segment Byte Public 'CODE'

        Public  decoder_header
decoder_header Label Byte
        Db      "DECODER1"              ;REV 1
        Dd      _init                   ;
        Dd      15 Dup (0)              ;Addresses of
                                        ;support
                                        ;routines

  Assume CS:HEADER_TEXT,DS:DGROUP,ES:Nothing,SS:Nothing

        Public  _htons
_htons  Proc    Far
        Push    BP                      ;
        Mov     BP,SP                   ;

        Mov     AX,[BP].PARM1           ;
        Xchg    AH,AL                   ;
```

```
        Pop     BP                          ;
        Ret                                 ;
_htons  Endp

  Assume CS:HEADER_TEXT,DS:DGROUP,ES:Nothing,SS:Nothing

        Public  _htonl
_htonl  Proc    Far
        Push    BP                          ;
        Mov     BP,SP                       ;

        Mov     AX,[BP].PARM1               ;
        Mov     DX,[BP].PARM2               ;
        Xchg    DH,AL                       ;
        Xchg    DL,AH                       ;

        Pop     BP                          ;
        Ret                                 ;
_htonl  Endp

HEADER_TEXT Ends
        End
```

STRUCTS.H

```
/*
 *      Copyright, Klos Technologies, Inc.
 *      All Right Reserved
 */


typedef unsigned char byte;
typedef unsigned short int word;
typedef unsigned long int dword;

#define ETHERNET                    0x00
#define TOKENRING                   0x08
#define ARCNET                      0x10
#define FDDI                        0x18
#define PPP                         0x20
#define SLIP                        0x28
#define MEDIA_MASK                  0x38

#define IEEE8022                    0x01
#define IEEE8022SNAP                0x02
#define DIX                         0x04

#define MEDIA_ETHERNET_8022         (ETHERNET+IEEE8022)
#define MEDIA_ETHERNET_8022_SNAP    \
        (ETHERNET+IEEE8022+IEEE8022SNAP)
#define MEDIA_ETHERNET_DIX          (ETHERNET+DIX)
#define MEDIA_TOKENRING_8022        (TOKENRING+IEEE8022)
#define MEDIA_TOKENRING_8022_SNAP   \
        (TOKENRING+IEEE8022+IEEE8022SNAP)
#define MEDIA_ARCNET                (ARCNET)
#define MEDIA_ARCNET_8022           (ARCNET+IEEE8022)
#define MEDIA_ARCNET_8022_SNAP      \
        (ARCNET+IEEE8022+IEEE8022SNAP)
#define MEDIA_FDDI                  (FDDI)
#define MEDIA_FDDI_8022             (FDDI+IEEE8022)
#define MEDIA_FDDI_8022_SNAP        \
        (FDDI+IEEE8022+IEEE8022SNAP)
#define MEDIA_PPP                   (PPP)
#define MEDIA_SLIP                  (SLIP)
```

```
#define BLUE    0x01
#define GREEN   0x02
#define CYAN    0x03
#define RED     0x04
#define MAGENTA 0x05
#define BROWN   0x06
#define WHITE   0x07
#define GREY    0x08
#define LTBLUE  0x09
#define LTGREEN 0x0a
#define LTCYAN  0x0b
#define LTRED   0x0c
#define LTMAGENTA 0x0d
#define YELLOW  0x0e

struct ethernet_header
{   byte destination[6];
    byte source[6];
    word type;
    byte data[];
};

struct token_ring_header
{   byte access_control;
    byte frame_control;
    byte destination[6];
    byte source[6];
    byte data[];
};
```

```
struct arcnet_header
{   byte source;
    byte destination;
    byte type;
};

struct fddi_header
{   byte frame_control;
    byte destination[6];
    byte source[6];
    byte data[];
};

struct PPP_header
{   byte direction;
    byte address;
    byte control;
    word type;
};

struct SLIP_header
{   byte direction;
};

struct sap_header
{   byte dsap;
    byte ssap;
    byte control;
};

struct snap_header
{   byte organization[3];
    word type;
};

struct protocol
{   struct protocol *next;
    char *protocol_name;
    word type1;       /* type field for DIX and SNAP */
    byte type2;       /* type field for 802.2 headers */
    byte type3;       /* type field for ARCNET */
    word type4;       /* type field for PPP */
    word type5;       /* to be defined */
    void (*show_line)();
    void (*show_packet)();
};
```

```
struct interface      /* REV 1 */
{   byte i_signature[8];
    struct protocol *(*i_initialize)();
    byte *(*i_sprintf)();
    void (*i_printf)();
    void (*i_format_protocol)();
    void (*i_format_protocol_line)();
    void (*i_format_raw)();
    void (*i_format_raw_line)();
    void (*i_set_color)();
    byte *(*i_falloc)();
    int (*i_open)();
    int (*i_read)();
    int (*i_write)();
    long (*i_lseek)();
    int (*i_close)();
    char *i_home_dir;
    int *i_current_level;
};

#ifdef INTERNAL_DECODER
char *sprintf();

extern char home_dir[];
extern int current_level;
#else /* EXTERNAL_DECODER */
extern struct interface *decoder_header;

#define sprintf (decoder_header->i_sprintf)
#define printf  (decoder_header->i_printf)
#define format_protocol         \
        (decoder_header->i_format_protocol)
#define format_protocol_line    \
        (decoder_header->i_format_protocol_line)
#define format_raw (decoder_header->i_format_raw)
#define format_raw_line         \
        (decoder_header->i_format_raw_line)
#define set_color  (decoder_header->i_set_color)
#define open       (decoder_header->i_open)
#define read       (decoder_header->i_read)
#define write      (decoder_header->i_write)
#define lseek      (decoder_header->i_lseek)
#define close      (decoder_header->i_close)

#define home_dir (decoder_header->i_home_dir)
#define current_level (decoder_header->i_current_level)
#endif

unsigned int htons();
#define ntohs htons
unsigned long htonl();
#define ntohl htonl
```

IP.H

```
/*
 *      Copyright, Klos Technologies, Inc.
 *      All Right Reserved
 */

struct arp_header
{   word type;
    word protocol;
    byte node_len;
    byte host_len;
    word operation;
    byte source_node_addr[6];
    dword source_host_id;
    byte target_node_addr[6];
    dword target_host_id;
};

struct arp_header2
{   word type;
    word protocol;
    byte node_len;
    byte host_len;
    word operation;
    byte source_node_addr;
    dword source_host_id;
    byte target_node_addr;
    dword target_host_id;
};

struct ip_header
{   byte version_length;
    byte type_of_service;
    word length;
    word id;
    word fragment_offset;
    byte time_to_live;
    byte protocol;
    word checksum;
    dword source_host_id;
    dword destination_host_id;
    byte options[];
};

struct tcp_header
{   word source_port;
    word destination_port;
    dword sequence;
    dword acknowledgement;
    word control;
    word window;
    word checksum;
    word urgent_ptr;
};

struct udp_header
{   word source_port;
    word destination_port;
    word length;
    word checksum;
    byte data[];
};

struct rip_entry
{   word address_family;
```

```
    word reserved1;
    dword ip_address;
    dword reserved2[2];
    dword metric;
};

struct rip_header
{   byte command;
    byte version;
    word reserved;
    struct rip_entry rip_entries[];
};
```

DEMO.C

```
/*
 *      This is a sample DECODER for Klos Technologies
 *      protocol analyzers.
 *      It decodes IP and ARP packets.
 *
 *      Copyright, Klos Technologies, Inc.
 *      All Rights Reserved
 */

#include "structs.h"
#include "ip.h"

void _loadds format_ip_line();
void _loadds format_ip();
void _loadds format_arp_line();
void _loadds format_arp();
```

```
/*                                                               */
/* This is a multiple protocol decoder. It supports both DOD */
/* IP and ARP.  Note how they are chained in the init()      */
/* routine.                                                      */
/* The last entry in the chain should contain a NULL next    */
/* pointer.                                                      */
/*                                                               */
/*   PPP packet type goes here ============================\\*/
/*                                                             ||*/
/*   ARCnet packet type goes here ==================\\      ||*/
/*                                                   ||      ||*/
/*   802.2 SAP type goes here ================\\    ||      ||*/
/*                                              ||    ||      ||*/
/*   Ethernet type goes here ==========\\     ||    ||      ||*/
/*                                      ||     ||    ||      ||*/
/*   Protocol Name goes here            ||     ||    ||      ||*/
/*                        ||            ||     ||    ||      ||*/
/*                        VV          VVVV    VV    VV    VVVV*/
struct protocol ip_protocol =
                  { 0, "Demo IP",  0x0800, 0x06, 0xf0, 0x0021,
                                    0, format_ip_line,
        format_ip };
struct protocol arp_protocol =
                  { 0, "Demo ARP", 0x0806, 0x00, 0xf1, 0x0000,
                                  0, format_arp_line,
        format_arp };

char yes[] = "yes";
char no[] = "no";

char *well_known_protocols[] =
{   "ICMP", "GGP", "TCP", "EGP", "IGP", "CHAOS", "UDP", "TP4"
};

byte protocol_lookup[] =
{   1, 3, 6, 8, 9, 16, 17, 29
};

char *hardware_types[] =
{   "Ethernet (10MB)",
    "Ethernet (3MB)",
    "Amateur Radio AX.25",
    "Proteon PROnet Rings",
    "CHAOSnet",
    "IEEE 802",
    "ARCNET"
};

struct protocol * _loadds init()
{
    ip_protocol.next = &arp_protocol;
    return (&ip_protocol);
}
```

```
void _loadds format_arp(arp, length)
    struct arp_header *arp;
    int length;
{   int i, j;

    set_color(LTGREEN, YELLOW);

    printf("DEMO DoD ARP:\n");
    i = htons(arp->type);
    if ((i > 1) && (i < 7))
        printf("Hardware type = %s\n", hardware_types[i-1]);
    else
        printf("Hardware type = %04x\n", i);
    printf("Protocol = %04x\n", htons(arp->protocol));
    printf("Node address length = %d, Host address length =
        %d\n",
                arp->node_len, arp->host_len);
    i = htons(arp->operation);
    if ((i < 1) || (i > 2))
    {   printf("Operation = UNKNOWN (%d)\n", i);
        return;
    }
    printf("Operation = %s\n", (i == 1) ? "REQUEST" : "REPLY");
    printf("Source node address: %n  Source host address: %i\n",
            arp->source_node_addr, htonl(arp->source_host_id));
    if (i == 1)
        printf("Target node address: UNKNOWN       Target host
        address: %i\n",
               htonl(arp->target_host_id));
    else
        printf("Target node address: %n  Target host address:
        %i\n",
               arp->target_node_addr, htonl(arp-
        >target_host_id));
}

void _loadds format_ip(packet, length)
    byte *packet;
    int length;
{   int i, j, k;
    struct ip_header *ip = (struct ip_header *)packet;

    set_color(LTGREEN, WHITE);

    printf("DEMO DoD IP:\n");
    printf("IP version: %d  IP header length: %d (32-bit
        words)\n",
           ((ip->version_length&0xf0)>>4),
           (ip->version_length&0x0f));
    printf("Type of service: %02x\n", ip->type_of_service);
    printf("Packet length: %04x  Packet ID: %04x\n",
           htons(ip->length), htons(ip->id));
    i = htons(ip->fragment_offset);
    if (i&0x8000)
        printf("Don’t fragment\n");
    else
        printf("More fragments: %s  Fragment offset: %04x\n",
                (i&0x4000) ? yes : no, (i&0x3fff));
    i = ip->protocol;

    for (j=0; j<sizeof(protocol_lookup); j++)
        if (protocol_lookup[j] == i)
            break;
    if (j != sizeof(protocol_lookup))
        printf("Time-to-live: %d  Protocol: %s  Header checksum:
        %04x\n",
               ip->time_to_live, well_known_protocols[j],
```

```
                htons(ip->checksum));
    else
        printf("Time-to-live: %d  Protocol: %d  Header checksum:
        %04x\n",
                ip->time_to_live, i, htons(ip->checksum));
    printf("Source host id:      %i\n", htonl(ip-
        >source_host_id));
    printf("Destination host id: %i\n",
           htonl(ip->destination_host_id));

    i = (ip->version_length&0x0f)*4;
    if (length <= i)
        return;

    packet += i;
    length -= i;

    printf("\n");

    switch (ip->protocol)
    {
    case 83: /* Vines IP */
        format_protocol(packet, length, 0xff00);
        break;

    default:
        format_raw("IP Data:", packet, length);
        break;
    }
}

void _loadds format_arp_line(b, arp, length)
    char *b;
    struct arp_header *arp;
    int length;
{
    set_color(LTGREEN, YELLOW);
    b = sprintf(b, "DEMO DoD ARP: (%04x) ", htons(arp-
        >protocol));

    switch (htons(arp->operation))
    {
    case 1:
        sprintf(b, "REQUEST from %i for %i",
                htonl(arp->source_host_id),
                htonl(arp->target_host_id));
        break;
```

```
    case 2:
        sprintf(b, "REPLY from %i to %i",
                htonl(arp->source_host_id),
                htonl(arp->target_host_id));
        break;

    default:
        sprintf(b, "UNKNOWN");
        break;
    }
}

void _loadds format_ip_line(b, packet, length)
    char *b;
    byte *packet;
    int length;
{   int i, j, k;
    struct ip_header *ip = (struct ip_header *)packet;

    set_color(LTGREEN, WHITE);
    b = sprintf(b, "DEMO DoD IP: %i -> %i ",
                htonl(ip->source_host_id),
                htonl(ip->destination_host_id));

    i = ip->protocol;
    for (j=0; j<sizeof(protocol_lookup); j++)
        if (protocol_lookup[j] == i)
            break;
    if (j != sizeof(protocol_lookup))
        b = sprintf(b, "%s: ", well_known_protocols[j]);
    else
        b = sprintf(b, "%d: ", i);
}
```

Appendix C - Crynwr Packet Driver Collection

This appendix describes how to use the Crynwr Packet Driver Collection provided with PacketView. The following information is provided as a quick reference. The entire contents of the files SUPPORT.DOC and INSTALL.DOC are available on the Crynwr Packet Driver Collection diskette.

Crynwr Software sells support to packet driver users.

This is what support includes:

- The assurance that the drivers will continue to be improved,
- New packet driver releases automatically mailed to you,
- Input into future packet driver developments.
- Answers to questions on the phone to one person or an alternate,
- Answers to questions emailed by anyone at your site.

Number of adapters	Price (year-long contract)
1-5	$50
6-64	$100
65-499	$100 + $1.50/adapter past 65
500-1499	$850 + $1.00/adapter past 500
1500-	$1850 + $0.80/adapter past 1500

Special pricing is available for special circumstances. Crynwr also sells support to vendors of hardware and software that use packet drivers.

We can accept checks and purchase orders. We accept orders via phone, FAX, or email. We're a small company, so checks are preferable. Prices subject to change without notice.

Crynwr Software
11 Grant St.
Potsdam, NY 13676
(315)268-1925 FAX: (315)268-9201
info@crynwr.com

Crynwr Packet Driver Installation

(excepts from the file INSTALL.DOC on the Crynwr Packet Driver Collection diskette)

All numbers in this appendix are given in C-style representation. Decimal is expressed as 11, hexadecimal is expressed as 0x0B, octal is expressed as 013. All reference to network hardware addresses (source, destination and multicast) and demultiplexing information for the packet headers assumes they are represented as they would be in a MAC-level packet header being passed to the send_pkt() function.

Using the packet drivers

The packet driver must be installed prior to use. Since each packet driver takes only a few thousand bytes, this is best done in your AUTOEXEC.BAT. Since the Ethernet boards typically have jumpers on board, the packet driver must be informed of the values of these jumpers (auto-configure is possible, but can disturb other boards). The first parameter is the software interrupt used to communicate with the packet driver. And again, because each board is different, the rest of the parameters will be different.

All parameters must be specified in C-style representation. Decimal is expressed as 11, hexadecimal is expressed as 0x0B, octal is expressed as 013. Any numbers that the packet driver prints will be in the same notation.

Before installing the packet driver, you must choose a software interrupt number in the range between 0x60 and 0x80. Some of these interrupts are used for other purposes, so your first choice may not work.

Running a packet driver with no specifications will give a usage message. The parameters for some packet drivers are documented below.

Most drivers can also be used in a PROM boot environment, see PROMBOOT.NOT for how to use -d and -n options for that purpose.

The -w switch is used for Windows. Install the packet driver before running MS-Windows. This switch does not prevent Windows from swapping your network application out of memory, it simply detects when that has happened, and drops the packets on the floor.

NOTE: Not all packet drivers listed below have been tested with PacketView. Please call Klos Technologies, Inc. customer support if you are having problems with a particular packet driver.

3Com 3C501

```
usage: 3C501 [-n] [-d] [-w] packet_int_no [int_no [io_addr]]
```

The 3C501 driver requires two additional parameters -- the hardware interrupt number and the I/O address. The defaults are 3 and 0x300.

3Com 3C503

```
usage: 3C503 [-n] [-d] [-w] packet_int_no [int_level(2-5)
           [io_addr [cable_type]]]
```

The 3C503 driver requires three additional parameters -- the hardware interrupt number, the I/O address, and the cable type. The 3C503 can be attached to thick or thin Ethernet cables, and the selection is made in software. The cable type parameter should be zero for thick, and one for thin. The defaults are 2, 0x300, and 1 (thin). The 3C503 uses shared memory whose address is set by jumpers, but the software can ask the board what the address is.

3Com 3C507

```
usage: 3C507 [-n] [-d] [-w] packet_int_no [int_no [io_addr
           [base_addr]]]
```

The 3C507 will determine its parameters by reading the board. The only time you would need to specify the parameters is when you have multiple 3C507s in the same machine. The 3C507 driver will use three additional parameters -- the hardware interrupt number, the I/O address, and the memory base address.

3Com 3C523

```
usage: 3C523 [-n] [-d] [-w] packet_int_no [int_no [io_addr
           [base_addr]]]
```

The 3C523 driver requires three additional parameters -- the hardware interrupt number, the I/O address, and the memory base address. The defaults are 3, 0x300 and 0xc000.

BICC Data Networks' ISOLAN 4110 Ethernet

```
usage: ISOLAN [-n] [-d] [-w] packet_int_no [int_no
           [base_addr]]
```

The BICC ISOLAN requires three additional parameters -- the hardware interrupt number and the memory base address. The defaults are 2 and 0xb800h.

D-Link DE-600

```
usage: DE600 [-n] [-d] [-w] packet_int_no
```

The D-Link Pocket Lan Adapter packet driver requires no additional parameters.

HP Ethertwist

```
usage: HPPCLAN [-n] [-d] [-w] packet_int_no [int_no
           [io_addr]]
```

The HPPCLAN driver requires two additional parameters -- the hardware interrupt number and the I/O address. The defaults are 3 and 0x300.

ICL EtherTeam16

usage: ETHIIE [-n] [-d] [-w] packetintno [intlevel [ioaddr
        [cabletype]]]

The ETHIIE driver requires three additional parameters -- the hardware interrupt number, the I/O address, and the cable type. The interrupt levels supported by the adapter are 5, 9 (2), 12 and 15. The Ethernet IIe can be attached to thick or thin Ethernet cables, and the selection is made in software. The cable type parameter should be zero for thick, and one for thin. With the Twisted Pair (TP) version of the adapter, you must set interface to the value 1 (thin).

The defaults are 9 (2), 0x300 and 1 (thin).

Please note, that the adapter can be used only in a 16-bit slot of your computer.

Intel EtherExpress

usage: EXP16 [-n] [-d] [-w] <packet_int_no> [<io_addr>]

The Intel EtherExpress packet driver has one optional parameter. The <io_addr> is only needed if there is more than one EtherExpress card in your system. Otherwise, the driver will search for adapter and get its parameters from it.

Multitech EN-301

usage: EN301 [-n] [-d] [-w] packet_int_no [int_no [io_addr]]

The Multitech driver runs the EN-301 cards. The Multitech driver requires two additional parameters, the hardware interrupt number, and the I/O port.

Novell NE1000

usage: NE1000 [-n] [-d] [-w] packet_int_no [int_no
        [io_addr]]

The NE1000 driver requires two additional parameters -- the hardware interrupt number and the I/O address. The defaults are 3 and 0x300.

Novell NE2000

```
usage: NE2000 [-n] [-d] [-w] packet_int_no [int_no
          [io_addr]]
```

The NE2000 driver requires two additional parameters -- the hardware interrupt number and the I/O address. The defaults are 2 and 0x300.

Racal-InterLan NI5010

```
usage: NI5010 [-n] [-d] [-w] packet_int_no [int_no [io_addr]]
```

The NI5010 driver requires two additional parameters -- the hardware interrupt number and the I/O address. The defaults are 3 and 0x300.

Racal-InterLan NI5210

```
usage: NI5210 [-n] [-d] [-w] packet_int_no [int_no
          [io_addr [base_addr]]]
```

The NI5210 driver requires three additional parameters -- the hardware interrupt number, the I/O address, and the memory base address. The defaults are 2 and 0x360 and 0xd000. Note that Racal-InterLan sets the default memory base to 0xa000, which is brain-damaged, because that area of memory is specifically reserved for video adapters, and in fact the EGA and VGA use it.

Racal-InterLan NI6510

```
usage: NI6510 [-n] [-d] [-w] packet_int_no [int_no [io_addr]]
```

The NI6510 driver has two additional parameters -- the hardware interrupt number and the I/O address. The defaults are 2 and auto-sense. These parameters do not need to be set unless the auto-sense routine fails, or otherwise disrupts operation of your PC.

Racal-InterLan NI9210

    usage: NI9210 [-n] [-d] [-w] packet_int_no [int_no [io_addr
        [base_addr]]]

The NI9210 driver requires three additional parameters -- the hardware interrupt number, the I/O address, and the memory base address. The defaults are 2, 0x360 and 0xd000.

Tiara Lancard

    usage: tiara [-n] [-d] [-w] packet_int_no [int_no [io_addr]]

The Tiara driver runs the Tiara LANCARD/E cards, both eight and sixteen bit cards. The Tiara driver requires two additional parameters, the hardware interrupt number, and the I/O port.

Ungermann-Bass NIC-PC

    usage: UBNICPC [-n] [-d] [-w] <packet_int_no> <int_no>
        <base_addr>

The UB NIC-PC driver requires two additional parameters, the hardware interrupt number, and the memory base address.

Western Digital WD8003 E EBT EB ET/A and E/A

    usage: WD8003E [-n] [-d] [-w] packet_int_no [-o] [int_level
        [io_addr [mem_base]]]

The WD8003E driver runs the Western Digital E, EBT, EB, ET/A, and E/A Ethernet cards. The WD8003E requires three additional parameters -- the hardware interrupt number, the I/O address, and the memory base address. The defaults are 2 and 0x280 and 0xd000. The wd8003 cards do not enable their memory until configuration time. Some 386 memory mappers will map memory into the area that the card intends to use. You should be able to configure your software to leave this area of memory alone. Also driver will refuse to map memory into occupied memory. The occupied memory test fails on some machines, so the optional switch "-o" allows you to disable the check for occupied memory.

Appendix D - Klos Technologies, Inc. Packet Drivers

Klos Technologies, Inc. makes two enhanced packet drivers available for use with ISDNView™. These packet drivers provide error information to ISDNView™, allowing a more complete view of the network. At this time, only two packet drivers are available with these extended capabilities. One for ethernet NE1000 and NE2000 (and compatible) boards, and one for Cimetrics ARS-20020 (and other COM20020 based) ARCNET boards.

ETHPD

ETHPD is an enhanced packet driver for NE1000, NE2000 and compatible ethernet adapters. It automatically detects the bus-width and memory size of the adapter.

Example:

```
ethpd [/p:nnn][/h:nn][/s:nn]
```

The optional switches allow you to select a configuration other than the default configuration for the packet driver.

Switch	Description
/p:nnn	Select the I/O base address for the ethernet adapter. The default I/O base address is 300 (hex). To use a different I/O base address, specify the address "nnn" as a HEXADECIMAL value.
/h:nn	Select the hardware interrupt request level for the ethernet adapter. The default hardware interrupt request level is 5. To use a different hardware interrupt request level, specify the level "nn" between 2 and F (inclusive) as a HEXADECIMAL value.

/s:nn Select the packet driver's software interface interrupt. The default software interface interrupt is 60 (hex). If this value causes a conflict with other software in your PC, select another value between 60 hex and 80 hex (inclusive). Specify the new value "nn" in HEXADECIMAL.

COM20020

COM20020 is an enhanced packet driver for SMC COM20020 based ARCNET adapters. To start the packet driver, simply execute COM20020 from the command line.

Example:

```
        com20020
[/a:nn][/p:nnn][/h:n][/s:nn][/r:n][/b:n]
```

The optional switches allow you to select a configuration other than the default configuration for the packet driver.

Switch	Description
/a:nn	Select the 8-bit network node address for the COM20020. If the board is compatible with the ARS-20020 board from Cimetrics Technology, the default is to use the network node address set on the SW2 switch. You can override the default setting by selecting this option where "nn" is the HEXADECIMAL value of the desired network node address. If the board you are using is NOT compatible with the ARS-20020, you MUST use this switch to set the desired network node address for your board.
/p:nnn	Select the I/O base address for the COM20020. The default I/O base address is 300 (hex). To use a different I/O base address, specify the address "nnn" as a HEXADECIMAL value.

/h:n Select the hardware interrupt request level for the COM20020. The default hardware interrupt request level is 5. To use a different hardware interrupt request level, specify the level "n" between 2 and 7 (inclusive).

/s:nn Select the packet driver's software interface interrupt. The default software interface interrupt is 60 (hex). If this value causes a conflict with other software in your PC, select another value between 60 hex and 80 hex (inclusive). Specify the new value "nn" in HEXADECIMAL.

/r:n Select the network speed of the COM20020. The default network speed is 0 (for 2.5 Mbps). Values for "n" are listed below:

Value	Network Speed
0	2.5 Mbps (default)
1	1.25 Mbps
2	625 Kbps
3	312.5 Kbps

/b:n Select backplane mode for the COM20020. To enable backplane mode, use "/b:1". To disable backplane mode (default), use "/b:0".