HP 4972A Protocol Analyzer

HP 18220SA
Protocol Interpreter
Development Environment

User's Guide

HEWLETT
PACKARD

Manual Part Number: 18220-99501
Microfiche Part Number:18220-98801

Printed in U.S.A. September, 1989
E0989

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

If your software application or hardware should fail, contact your local Hewlett-Packard Sales Office listed in the protocol analyzer operating manual.

Colorado Telecommunications Division
5070 Centennial Boulevard
Colorado Springs, CO, 80919-2497

Contents

Getting Started

Note: The HP 182205A Decode Development System requires Revision B.03.00 of the HP 4972A Operating System.

To use the HP 4972A application development environment, you must first turn your HP 4972A i Pascal workstation. The HP 98617A Pascal Language System (also required to use the HP 18220 has detailed instructions for this. The following are simple tips to help reduce set-up time and enl performance of your system.

Setting Up the Pascal Workstation

Your HP 4972A already has much of the Pascal Workstation System (PWS) software installed o hard drive. Volume #11 should contain the files SYSTEM_P, STARTUP, INITLIB, TABLE, an AUTOKEYS. However, these files are for Revision 3.01 of the PWS, and you will need to replac some of them with Revision 3.22 versions.

To do this, put the "BOOT:" disc from your 3.22 PWS into the floppy disc drive of the 4972A. Cyc power to the 4972A while holding the space bar down. Keep holding the space bar down until the message "SEARCHING FOR A SYSTEM" appears at the bottom of the screen. Then enter "2F from the keyboard. The system should continue booting from the floppy disc. When the boot has finished, the system will prompt you for the correct date and time. Enter these, and the PWS command line will be displayed.

At this point you must remove the "BOOT:" disc and insert the "ACCESS:" disc. Then press "F" t the FILER. Once the FILER has loaded you may again swap discs, removing "ACCESS:" and ins "BOOT:".

Now remove files SYSTEM_P, STARTUP, INITLIB, findSYS.CODE and AUTOKEYS from #1 Krunch the volume, then (in the Filer) filecopy SYSTEM_P, STARTUP, and INITLIB from #3 t #11. IT IS NEITHER NECESSARY NOR DESIRABLE AT THIS POINT TO REPLACE TH TABLE PROGRAM ON #11.

Next, remove the "BOOT:" disc and re-boot by pressing < Shift > < RESET >. The system sho now boot off of the internal hard disc and prompt you for the date and time again. Hit < Return twice to get to the PWS command line again.

ısert the "ACCESS:" disc and press "F" to load the FILER. Now copy the FILER, EDITOR, IBRARIAN, and MEDIAINIT.CODE files from #3 to #13. Then replace the "ACCESS:" disc wi ıe "CMP:" disc and copy the COMPILER from #3 to #13. Finally, replace the "CMP:" disc with th ASM:" disc and copy the ASSEMBLER and DEBUGGER files from #3 to #13. Your hard disc n ɔntains all of the PWS system files you will need.

.e-boot your system once more. From the PWS command line, press "W". The system should displ list of system files and their locations. All of the files except the LIBRARY file should be prefixed HARDSC:" indicating that the system has found them on the hard disc. If so, you are now ready to ɔad the HP 18220SA files onto your hard disc.

.oading the HP 18220SA Files

'he HP 18220SA disks contain several files which will help you to create HP 4972A applications. T re:

DISK 1	DISK 2
AUTOKEYS	EXPORTS.TEXT
indSYS.CODE	XREF.TEXT
.IBRARY	
:TABLE72.TEXT	
ink_72.TEXT	
;YS_72.CODE	
yz_body.TEXT	
yz_body.CODE	
yz_main.TEXT	
yz_main.CODE	
ink_xyz.TEXT	
XYZ0.A	
ɔuild_IL.TEXT	

nsert the HP 18220SA disc into the floppy drive and copy AUTOKEYS and findSYS.CODE to #1 These will allow you to select between the PWS and the HP 4972A instrument code when the unit i ɔowered on.

The remaining files should be copied onto #13 for use in developing your applications. You can us he wildcards (=,$) to speed up the copy process. A brief explanation of those files follows:

:IBRARY

Contains library routines needed for compilation. YOU MUST NOT USE THE LIBRARY FILE WHICH COMES WITH THE PWS SOFTWARE!

TABLE72.TEXT	This is a modified version of the Version 3.22 CTABLE program which comes with your PWS software. If you need to modify your system configuration, you should modify this file and compile it, then copy the result into #11:TABLE
link_72	This is a stream file which will combine the "library.CODE" file on #11 with the "HP 4971S.CODE" file on volume #12 into a file called "LPA.CODE" on volume #13. This allows you to invoke the analyzer software at any time by simply executing #13:LPA.
SYS_72	This file contains all of the exported modules from the HP 4972A base system. It must be "searched" by your applications files for them to compile.
XYZ_body.TEXT	This is an empty decode shell file which can be modified to create your custom decode. There will be one "body" file for each protocol header decoded.
XYZ_body.CODE	The code file generated when xyz_body.TEXT is compiled.
XYZ_main.TEXT	This is an empty application shell file which can be modified to create custom application. When used in conjunction with xyz_body, it can cr an entire decode application. Only one "main" is required for each prc decode stack.
XYZ_main.CODE	The code file generated when xyz_main.TEXT is compiled.
link_XYZ.TEXT	This is a sample stream file which invokes the LIBRARIAN to link the xyz_body and xyz_main CODE files together into XYZ0.A, an applica file.
XYZ0.A	Application file created by link_xyz.
build_IL	If your HP 4972A has Option #002 (remote interface) it will be neces to add two modules to your INITLIB file on #11. This stream file is provided to do that for you. Simply insert the ACCESS: disk from you PWS software and Stream (S) #13: build_IL. If the stream file fails fc some reason, consult your Procedure Library manual for instructions adding "RS232" and "DATA_COMM" to your INITLIB.
EXPORTS.TEXT	This file contains all of the data types, constants, variables, and proced exported from the base system. It can be used in conjunction with XREF.TEXT to identify useful items already implemented. There is or no documentation associated with this file, so the items contained should be used only as a last resort.

:REF.TEXT This file contains a cross reference for the items contained in EXPORTS.TEXT.

\ Typical Development Cycle

Jow that you have access to the PWS and the HP 18220SA files, you can start developing applicatic
The process for this is fairly simple:

1. Edit the source file shells xyz_body and xyz_main to add the custom functions.

2. Compile the files. It is always advisable to compile in reverse hierarchical order. Thus xyz_body should be compiled first and xyz_main should be compiled second.

3. Use the link_xyz stream file to link the xyz_body and xyz_main files together into XYZ0.A

   NOTE: Whenever "xyz" is encountered, it should be replaced by the decode header title.

4. Invoke the analyzer software by running LPA.CODE (created in step 3).

5. Load your application from the disc functions menu.

6. Test your application.

7. Return to the PWS environment by pressing < Shift > <USER>. On an HP 4971S, use < Shift > key in conjunction with the blank key in the upper right-hand corner of the keyboard.

8. Return to step 1.

Decode Guidelines

The Pascal Workstation Operating System in conjunction with the Lan Protocol Analyzer Decode Platform make it possible for you to develop protocol decode application programs for the HP 497 These decode applications will enhance the analysis capabilities of the HP 4972A by presenting protocol information in an easy-to-read format.

This chapter describes in general terms the attributes of a decode application. It is not intended to a complete and exhaustive definition since every protocol will have its own unique needs. It is a se guidelines for the minimum credible solution, to which additional features can be added as those n become evident.

Decode Display

Layers

For the purposes of this manual it is assumed that the protocol suite may be decomposed into functional layers along the lines of the OSI model. Although most protocols were defined before t OSI model, it is generally possible to "force fit" them into at least some subset of the OSI layers.

Within the Select Format menu, the layers are numbered 1 through 7. This is done primarily to pr the user with an easy reference. If the user has an unusual protocol stack, the decodes will still be called in the order that they occur in the frame. For example, IP is usually considered to be a leve three protocol. Suppose the network was running IP on top of the X.25 protocol. IP would be cal level four.

Overall Display

Each decoded layer is preceeded in the margin by the header name followed by a colon. Then, all header fields are shown in the order that they occur within the frame. If the decode fields will not on one line, the display is continued on the next line beginning with a colon and then followed by t remaining fields. A header display should consume as many lines as desired while still maintainin concise format. The displayed header might look something like the following:

```
leader: Field 1      00-00-00-00    Field 2      65535    Field 3      255
      : Field 4      Broadcast      Field 5      10       Field 6      00-00
      : Field 7      90 Unknown     Field 8      0
```

The title "Header" is the name of that particular header. The header name may be between one an fifteen characters long, and should only appear on the first line of the header display.

In the above example, the Field # represents the label that describes the data in the field. The abo example shows three fields per line, but the display may contain as many fields as will fit only onto line while still maintaining clear field labels. The data should be shown in a format that is the most meaningful. In the above example, fields 1 and 6 are shown in hexadecimal format, fields 2, 3, 5, 7, 8 are shown in decimal, and fields 4 and 7 both contain neumonics.

A header may be incomplete as a result of lost data, etc. What if there are not enough bytes in the frame to display the complete header? Only the fields that are complete should be displayed. For example, suppose the header displayed above contained only 12 bytes (assume that its full 16 bytes shown above) which is just enough to consume fields 1 through 5. The header would then be displ as follows:

```
Header: Field 1      00-00-00-00    Field 2      65535    Field 3      255
      : Field 4      Broadcast      Field 5      10
      :                          ***** Incomplete Header *****
```

Note that the display was cut short after field 5. Field 6 was not displayed on the second line as be The user must be notified that the header was incomplete. Therefore, an "Incomplete Header" message is put up on the very next display line. That message has a margin followed by a colon, an preceeded and trailed by five asterisks.

For another example, suppose there were only two bytes of data available in the header. Notice th field #1 requires four bytes of data. The header display would look as follows.

```
Header:                          ***** Incomplete Header *****
```

The user could turn "On" the SHOW HEX BYTES column (which will be explained later) to view two bytes of data.

The field labels should be targeted toward the decode user. If the user is familiar with the protoc labels may be abbreviated to the extent that their meaning is still clear.

Note that the data is right justified following the labels. This is done to give the display a "clean" l thereby allowing the user to become familiar with field data by position. The field labels should li as much as possible, giving the display a "column" appearance. For some headers, columns may n practical. The display should still maintain a degree of neatness.

A header display should never show more data than the previous header's length field indicates. Th lisplay should be cut off at that length.

Screen Attributes

Various attributes are used in the display to aid in finding information and locating error condition: The decode display should adhere to the screen attribute conventions that have been established. There are currently four attributes in use.

Normal Attributes

This attribute is used for the margin header name, the colon, the incomplete header message, and the data display if the data is nc error. This will be the prevalent attribute.

Half Bright Inverse Attribute

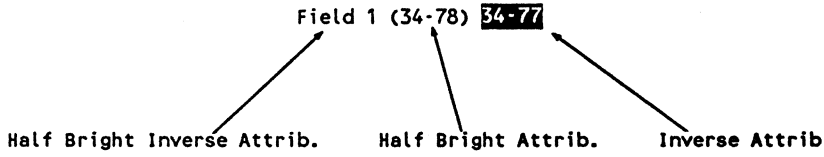
This attribute is used to display all of the field labels. In the abo display examples, the labels Field # will be in half bright inverse (Note: Within the system, the actual attribute used is the half bri inverse plus the underline attribute to keep the field labels from running together.)

Inverse Attribute

This attribute is used to flag an error in the data field. It was ch because it stands out. Example error conditions include: a bad checksum, a length field that is too long or too short, an invalid t or code field, etc. (Note: Within the system, the actual attribute used is the inverse attribute plus the underline attribute to keep field labels from running together.)

Half Bright Attribute

This value is used in the case of an incorrect checksum. The val that is in the checksum field is shown in inverse video as previou described. It may then be beneficial to show what the correct checksum value should have been. This correct value is shown ir half bright attribute preceeding the incorrect value. An example bad checksum would look as follows:



If the checksum is good, the value in parenthesis would be replaced by the word "Good" in normal attribute: Good 34-77. If the checksum field is not used, the value in parenthesis would be replace the word "Unused": Unused 00-00.

Within the Select Format menu, you can alter the decode display slightly. These changes will be described later in this chapter.

User Lists

Definition

It is often beneficial to be able to keep a list of addresses, services, etc. that are currently on the network. Each protocol may have different address formats. Therefore, provisions have been ma within the HP 4972A protocol decode platform to allow new lists to be added with each new decoc

What constitutes a valid list? A list may be added for any address field (this includes hosts, ports, sockets, etc.) or protocol field within the header. In this manual, protocol field is defined to be a fi which accesses a higher layer. For example, the IP protocol has a field which identifies the protoc that rides directly above IP (TCP, UDP, etc.).

What are the benefits of such a list? Within the decode display, the address or protocol fields can shown in a neumonic representation, thus making it much easier to interpret that field. If the lists not provided, the user would have to convert the data into a meaningful name. If "User Labels" is selected in the "SUBSTITUTE LABELS" column within the Select Format screen, the labels will b displayed in place of the data values provided an entry exists for that value. Each decode may hav more than one list. In theory, a list could be added for each address or protocol field.

It may be more practical for a decode developer to add an internal user list rather than one that appears as a softkey choice. This will greatly decrease the effort required. If the developer does choose to add an external list, the information that follows should prove useful.

Softkeys

The lists may be accessed through the Setup Analyzer softkey in the top level menu. When "Edit I is pressed, a menu of softkeys containing all of the lists that are currently loaded is displayed. The Physical Address List and the Ethernet Type List will always be resident. When each decode application is loaded, its corresponding lists will be loaded also. When the application is unloaded, lists will be removed. If more than one application is loaded, there may be quite a few lists. By pressing the "Other Choices" softkey, the remaining lists will be displayed. The user may enter a li simply pressing its corresponding softkey. The code for the lists is in a separate file from that of th application. That list file must be present on the same volume as the application file in order to be loaded.

.ist Display Format

. typical list will contain four elements: a list name, a column containing the entry number, a colun ontaining the name, and a column containing the address or data value.

'he list will appear as follows:

.ist Name: Example_List

Entry #	Name	Address
1	Example_Name	00-00-00-00

The List Name header, column headers, and entry numbers should all appear in normal attribute. .ist Name and the name and address fields should all appear in half bright inverse attribute except he field where the cursor currently resides. That field should be shown in full bright inverse attrib For a better example of the list attributes, please refer to the EtherType List, which resides in the I 4972A.

The softkeys that are provided within the list menus are as follows:

Softkey #1: Insert Address

This softkey allows the user to insert an entry in the list. The softkey label will change with each list. For example, in the EtherType list, this softkey reads "Insert Type".

Softkey #2: Delete Address

This softkey allows the user to delete an entry from the list.

Softkey #3: Sort Addresses

This softkey allows the user to sort the list by name or address. The softkey label will change with each list. For example, in the EtherType list, this softkey reads "Sort Types". Within this menu, two more softkeys will appear: "Sort by Name" and "Sort by Address".

Softkey #4: Search for Address

This softkey allows the user to search for a particular item. Within this menu, three more softkeys appear: "Search for Address #", "Search for Name", and "Search for Address". Again, the label "Address" may change depending on the type of list.

Softkey #5: Select Format

This softkey is present only when the cursor is in the data (address, type value, etc.) field. It allows the user to select how that field will be displayed. Within this menu, one to four softkeys will appear: "Decimal", "Hex", "Bin LSB Right", and "Bin LSB Left". (The last two softkeys represent Binary Least Significant Bit Right and Left, respectively.) In the case of some fields, it may not make sense to represent the data in one of these formats. In that instance, those formats are deleted from this list. Note that in the preceeding example, the data is shown in hexadecimal.

Softkey #6: Reset List

This softkey allows the user to set the list back to the original state in which it contains only the pre-defined values. Within this softkey menu, the user will be given the choice "Yes" or "No".

Please refer to the lists within the HP 4972A to view the softkey menus, thereby making some of tl elements discussed here much easier to picture.

File Extension

Each file type within the HP 4972A has a different extension. For example, data files end in '0.D'. '.D' tells the analyzer that a particular file is a data file. Each list in the system must also have its (extension since a list constitutes a file type also. The possible choices are all letters of the alphabe both uppercase and lowercase.

Some of the letters have already been used and therefore, since they have reserved a specific file t they cannot be reused. These extensions are: A,a,C,c,D,d,e,F,i,M,m,N,n,o,P.p,q,s,T,t,X,x.

Any other letter may be used for a new file type. In general, it is best to use upper case letters for multi-object files and lower case letters for single-object files.

Run Application Screen

Normally, an application is entered through the Run Application softkey in the top level menu. W this key is pressed, the user is given a softkey list of the applications that are currently loaded in th system. Upon selecting one of those softkeys, the application code is evoked, and program contro to the application.

The decode applications operate differently. When the application is loaded, the code becomes resident in the system and is accessed through the Examine Data menu. Therefore, the Run Application softkey need not be pressed unless the user wishes to unload the application. If the u

oes enter the application through the Run Application menu, a message should be provided giving ıort explanation of how to use the decode application. Thus, the Run Application screen is used as elp menu.

)ecode Screen Format

'he decode application screen contains the following elements:

- The application name and revision code element
- The protocol decodes that have been successfully loaded with the application
- A brief summary on how to use the application

'he decode application screen format should appear like this: (In the following example, the TCP/ ecode application is used.)

```
          TCP/IP Protocol Interpreter Application A.00.00

The following protocol decodes have been successfully loaded with
this application:

                                DOD IP
                                ICMP
                                ARP
                                RARP
                                TCP
                                UDP

     The decodes may be utilized through the  Examine Data  menu.

Press  <Examine Data>  then  <Select Format> and turn on the
appropriate header displays.  You may force each level to be
decoded as a specific protocol by entering the 'Protocol' column
and then using the softkeys to select the protocol of your choice.
A 'Default' entry in this column allows the analyzer to decode the
level.
```

Select Format

The Select Format screen allows the user the flexability to define how the decode should be displa
This functionality allows features such as the suppression of certain headers, a decoded header to b
shown in hex format, the selection of a summary or detailed format, the forcing of a certain layer t
decoded in a certain way, and the presentation of names in place of addresses or protocol fields. I
short, the user can tailor the decode to meet specific needs. This increased functionality is describe
the sections that follow.

The Header Column

The first column in the Select Format screen is labeled 'Header'. This column represents the layer
number corresponding to the seven layer OSI model. As described earlier, these layers are merely
reference, and need not match the protocol at hand. In other words 'HEADER 1' refers to the fir
decode (corresponding to the physical header), 'HEADER 2' refers to the second decode, (usually
corresponding to the link level header), and HEADERS 3 through 7 refer to the third, fourth, etc.,
protocol decode called. For example, suppose that the user decided to turn the HEADER 3 displa
off. This would result in the absence of the display of the third decode called, regardless of which
protocol that is.

The Select Format screen is part of the HP 4972A base system. This code itself is not modified wh
new application is added. However, the decode application itself must adhere to the format that th
user has selected.

The Display Column

The second column in the Select Format Screen is labeled DISPLAY. The softkey choices while w
this column are "Off" and "On". If the user selects "Off", that header will not be shown in the displ
The header will still be decoded, however.

This functionality allows the user to suppress certain headers from the Examine Data display. Thi
may be helpful, for example, if the user wishes to view only one header and, as a result, see more
frames on the display at one time. To prevent a frame from disappearing from the display, one le
must always be turned on.

The Detailed/Summary Column

In certain cases, the user may wish to view a scaled-down form of the header and scan for certain l
fields. For this purpose, a summary display option was added. The summary is defined as a one-li
synopsis of the header, and should contain a few key header fields. The summary format lets the u
find information without having to go through the entire header and also makes it possible for mo
frames to fit on the display. Within the "DETAILED/SUMMARY" column, the user can select e
"Detailed Display" or "Summary Display".

he fields displayed in the summary format are not selected by the user. They are defined by the pplication programmer. In some cases, a header may be so short that a summary need not be rovided. In these instances, the detailed header is displayed regardless of what the user selected in ıe Select Format menu.

'he Protocol Column

'he PROTOCOL column allows the user to select how each header should be represented. Under ormal conditions, the 'DEFAULT' value will be used. This allows the analyzer to decode the fram 'he user may force a layer to be decoded as a certain protocol by selecting one of the softkey choic: ;ach decode that is currently loaded in the system will appear on a softkey.

'he Hexbyte Column

'he user may wish to view the decoded header bytes in hex format as well as in detailed format. B: ntering the SHOW HEX BYTES column and turning the hexbytes "On", this function will be ıccomplished. Note: the hexbytes will be displayed only if that particular layer is turned "On" (the)ISPLAY column) and a decode application for that particular header is loaded in the system. An :xample display for a hypothetical 16 byte header might look as follows:

```
leader: 00-00-00-00-FF-FF-FF-FF-FF-FF-FF-0A-00-00-5A-00
      : Field 1      00-00-00-00   Field 2      65535   Field 3      255
      : Field 4      Broadcast     Field 5      10      Field 6      00-00
      : Field 7      90 Unknown    Field 8      0
```

The header name (in the margin) only appears on the first line and is followed by a colon just like t decoded header lines. If the header is longer than 20 bytes, the hex byte display will continue on th next line.

The Substitute Labels Field

Through the use of User Lists, the user can assign names to addresses or protocol fields. It is beneficial in many cases for these names to replace the data in the examine data display, thereby s the trouble of having to convert that data into a meaningful name. This is where the SUBSTITUT LABELS field comes into play.

In this field, the user can select either "User Labels" or "Data Values". If "User Labels" is selected, application accesses the list to see if a valid name has been defined for that data value. If a name c exist, the data value is replaced by the name in that field.

HP 4972A Application Interface

An HP 4972A application is a seperately compiled code module that can be dynamically loaded and linked in with the running HP 4972A program. When the user loads an application file, the HP 4972 uses the segmenter to load and link in the application code module. The application module has access to any procedures, functions, type declarations, and constants that are exported from modules included in the application module's import statement. The base system can access procedures and functions exported from the application module only through the system calls FIND_PROC(string), EXISTS_PROC(segment_proc), and CALL(procedure_variable, parameters).

Base System - Application Interface Requirements

Applications for the HP 4972A must reside in files ending in "0.A". Zero is the sequence number. "A" indicates the file is an application file.

The file name for the application, exclusive of the "0.A" suffix, must match the module name of the application. For example, the TCP/IP decode application is in a file called "TCPIP0.A". The module containing the application is declared as "MODULE tcpip". This is because the base system must use the module name concatenated with the procedure name in order to call an application procedure. The base system uses the file name as the module name when it does this. The file name must be in uppercase, but this is not necessary for the module name.

The procedures that the base system expects every application to export are outlined below along with their respective descriptions. These procedures must be exported from every application.

Procedure Initialization(VAR init_error : INTEGER)

This procedure will be called immediately after the HP 4972A loads the application. This procedure should initialize all application globals and attempt to allocate heap space (if needed) for large data structures. The variable parameter "init_error" should return 0 if initialization was successful, and a non-zero error code otherwise. If a non-zero code is returned, the base system will call unload to immediately unload the application.

Error code values :

= no error, initialization successful

= application already loaded

= insufficient heap memory for application

= insufficient global memory for application

= maximum applications already loaded

= some or all supplemental application files not found

= unresolved application references

100 = unable to load application due to software incompatibility *

1xx = I/O error, xx = IORESULT.

The initialization procedure will only have to return error codes 0, 2, 5, 6, and 1xx. Error codes 1, 3 and 100 are detected by the base system. Error codes greater than 100 are returned when an error detected via the Pascal Try/Recover mechanism. The recover statements should assign 100+IORESULT to the error code.

Procedure get_softkey_label(VAR sk_label : softkey_label)

Called by the base system to retrieve the softkey label that will appear for this application in the Application menu. The two lines of the softkey label can be up to nine characters each and both should not be blank.

Procedure entry_point

Called by the base system when the user runs this application within the Run Application menu. Entry_point is currently written as a shell to just call the main_application_procedure. It has a Try/Recover block to trap any fatal errors that occur within the application. It is advisable to leave entry_point unmodified and to just write the application within main_application_procedure. Main_application_procedure is where the actual application code resides. Decode applications wc slightly differently, as described in the Decode Application Outline section.

Note: This error usually occurs when external references cannot be resolved.

rocedure unload

alled by the base system when the user selects Unload Application. Must deallocate any heap emory allocated by the application. Also must do any necessary cleanup for the application. After is procedure is called, the base system will unlink and unload the application code module for the ase system.

The Decode Application

his chapter describes the procedures that appear in the application and perform the actual decodin nd display creation. It is presented in five parts:

1. The Modules
2. The Procedures
3. The Display Creation
4. System Procedures and Constants
5. Helpful hints for writing the decodes

The Modules

or each decode that you wish to include, you must create a separate module. Just as with the main pplication module, decode module naming is critical. This is because the base system can only call our decode if it is able to find modules and procedures of certain names.

he convention for module naming is fairly straightforward. The module name must be fifteen haracters or less and must match exactly the name of the protocol as defined by the previous layer. ou are designing a layer three protocol and you want to be called right after the level two decode i he base system, your module name must match the entry in Ethernet Type list or the 802.2 SAP lis our protocol. If you are designing a layer five protocol to run on top of TCP/IP, your module nam vould have to match the name of the corresponding port number in the TCP port list.

he Procedures

he decode procedures must be EXPORTED from the decode module. They are called as follows:

```
c_decode(  frame_number          : INTEGER;
           header_start          : INTEGER;
           current_level         : INTEGER;
           VAR protocol_label    : prot_name_type;
           VAR next_protocol     : prot_name_type;
           VAR next_start        : INTEGER;
           VAR protocol_end      : INTEGER );
```

x is the protocol name, which may be up to 15 characters long and must match the name of the de nodule. 'prot_name_type' is of type string [15] and is defined within the system.

For example, an XNS layer 3 decode has module name "xns_idp" and exports procedure "xns_idp-decode".

The parameters represent the following:

frame_number: The number of the frame currently being decoded and displayed.

header_start: The starting byte of the current header being decoded and displayed.

current_level: The level (1 thru 7) that the current header occupies in the frame.

protocol_label: A character label that contains the name of the protocol header and may be up to 15 characters in length. This label will be used in the display.

next_protocol: A character label that contains the name of the protocol header that lies on top of the current header, which is passed back to the calling environment s that the next layer can be called. If the next protocol cannot be found from the current layer, this parameter should return the value 'UNKNOWN'.

next_start: This value is equal to the current header_start plus the length of the current header. It is passed back to the calling environment, telling that environment a which byte in the frame the next header starts. Next_start should equal the header_start plus the number of bytes displayed.

protocol_end: This value is passed in by the calling enviroment. It is the maximum ending p for this header. If the decode application calculates a header length that exce this value, it must cut it's display short so that protocol_end is not exceeded. Next_start must also be cut accordingly. If the current level has a total length it then must calculate a new protocol_end, which will be returned to the callin environment as the maximum ending point for subsequent levels. This value i

used to calculate pad bytes. If the current level does not have a total length field, protocol_end should return the value that was passed in.

he Display Creation

he decode procedure is responsible for decoding and displaying the header. The display is not actu: nt to the screen from the application, it is merely created here. The system contains a display anager, which is responsible for writing the display lines and attributes to the screen, and is also sponsible for scrolling the frames, etc.. The application programmer communicates with the displ anager through the following procedures:

```
dspmgr_write_string( string );
dspmgr_write_attrib( screen attribute,
                     starting attribute column,
                     ending attribute column );
```

he first procedure sends the string to the display manager. The parameter passed is a string of aximum length 80 (the number of columns on the screen). The second procedure is used to send ttributes to the display manager for the string just sent. The parameters include the attribute, the arting screen column for the attribute, and the ending column for the attribute. The programmer rst sends a string to the display manager, and may then write as many attributes as he/she wishes f at string. This process is repeated for each string that the programmer desires to display in the urrent header. The allowable attributes are exported from the system module 'lan_lib', and are as llows:

```
normal_attrib
inverse_attrib

blinking_attrib
underline_attrib

half_bright_attrib
half_inverse_attrib
```

Any combination of these attributes may be used. For example, the current decode application lisplays the field labels in half bright inverse. To prevent the labels from running together on the creen, the underline attribute is also used, giving that attribute the value: half_inverse_attrib + nderline_attrib. A bad field value was given the attribute: inverse_attrib + underline_attrib. The d n the fields was displayed in normal_attrib. If normal_attrib is desired, no attribute need be sent t he display manager. This attribute is the default.

The system module 'selfrmt' contains a record structure called 'format', which holds the display for hat the user has selected. The programmer may need to access this structure and adjust the displa accordingly. For example, if the hexbyte column is turned 'On', the application should call a routin hat displays the headers in hex format. (This routine will be described later.) Also, the user may s

ether to show a detailed display of the header or a summary display. If the application programme ;hes to provide a summary display, the code must first check the detailed/summary format variable decide how the header should be displayed. In addition, the programmer must check to see wheth ieader display is turned on or not.

is 'format' structure is EXPORTED as follows:

(PE

```
v3_header_type = RECORD
                    display_on          : BOOLEAN;
                    hexbytes_on         : BOOLEAN;
                    summary_on          : BOOLEAN;
                    nbytes              : INTEGER;
                    name                : PROT_NAME_TYPE;
                END;

v3_headers_format_type = ARRAY [ 1..7 ] OF v3_header_type;

v3_frame_format_type = RECORD

hdrs            : v3 headers_format_type;
subs_labels     : BOOLEAN;              { Substitute labels }
timestamp       : timestamp_type        { Timestamp selection }
time_from_frm   : INTEGER               { Time from which frame }
base_high_int   : INTEGER               { Used to hold the base }
base_low_int    : INTEGER               { timestamp }
filter_on       : BOOLEAN               { Filter format on or off}
filter_to_use   : filter_display_type;  { Which filter format }
data_on         : BOOLEAN               { Whether datafield on }
data_format     : datafield_format_type; { Hex, Char, Hex & Char }
data_code       : char-mode type;       { Which data code type }
offset_in_bytes : BOOLEAN               { or after headers }
data_start      : INTEGER               { First byte of data }
                END;
```

AR
format : v3_frame_format_type;

he application programmer will only be concerned with:

```
format.hdrs[current_level].display_on      { Should display be called? }
format.hdrs[current_level].hexbytes_on     { Should hexbytes be displayed? }
format.hdrs[current_level].summary_on      { Summary or detailed display? }
format.subs_labels { Whether to display user labels or data values }
```

OTE: The programmer should check these values, but not alter them!

he HELPFUL HINTS section of this chapter will describe a suggested code format for the decode pplications.

ystem Procedures and Constants

he system will call xx_decode. Within this call, the application programmer is responsible for ecoding and generating the display strings. The programmer must be careful to ensure that the roper parameters have been set before returning to the calling environment.

/hile within the application, the programmer may wish to call some system routines. The following st of the routines that may be useful:

```
add_decode( prot_name : prot_name_type;
            prot_layer : INTEGER;
            VAR successful : BOOLEAN );
```

his procedure should be called by your application "initialization" routine for each decode you wish ld. Parameter "prot_name" is the name of the decode, and should match exactly the module name at decode. The "prot_layer" is the layer of the OSI stack where the protocol usually resides. The riable "successful" is returned to indicate if the protocol was successfully added or not. This call ould only fail if there is not enough memory in the system data structure to any more protocols.

```
remove_decode( prot_name : prot_name_type;
               prot_layer : INTEGER;
               VAR successful : BOOLEAN );
```

his procedure should be called by your application "unload" routine for each decode that you eviously added. This will remove that decode from the system decode list. Parameters are the san for add_decode.

```
splay_hexbyte_hdr( frame_number          :INTEGER;
                   header_start          :INTEGER;
                   number_of_byte        :INTEGER;
                   protocol_name         :prot_name_type);
```

ormat.hdrs[current_level].hexbytes_on is equal to TRUE, the above procedure should be called. It
l display the header bytes in hex. The parameter number_of_bytes should equal the total number c
es in the header. The protocol_name will be displayed as a label in the margin, which will be
ncated to five characters.

```
et_data_bytes(   frame_number            : INTEGER;
                 header_start            : INTEGER;
                 number_of_bytes_to_get  : INTEGER;
                 VAR put_bytes_here      : string_80;
                 VAR out_of_data         : BOOLEAN );
```

iis system routine is exported from the module 'newbuff'. You pass the frame number, starting byt
d number of bytes to get, and it returns the bytes in a string. If the boolean is set, there are not
ough bytes in that frame in the buffer; however, whatever bytes were available will be returned in t
ring variable. Using this routine, the user pulls the header bytes out of the buffer to be decoded an
splayed.

```
get_unmasked_hex_string( VAR data_byte_string    : string
                         VAR hex_string          : string );
```

his procedure takes a string of bytes (data_byte_string) and converts the string to a string of
exadecimal characters which is returned in hex_ string. The hex characters are separated by a '-'.
rogrammer must ensure that the hex string parameter is long enough to handle the returned string
or example, if data_byte_string is 2 elements long (say, 2 and 5), then hex string will read '02-05', a
ıust be 5 characters long. 'string' refers to a string of any length, which the programmer must defir
'his procedure may come in handy when displaying the data for certain fields in hex format.

)ther routines which may prove helpful include the following special routines: STRWRITE, ORD,
;IT_SET, BINAND, etc.. These system routines are documented in the Pascal Language Referenc
nd Procedure Library Manuals included in the Pascal Language System.

'he display manager procedures dspmgr_write_string and dspmgr_write_attrib were explained in th
)revious section.

veral types have been defined, which may also come in handy to the application programmer. The
)es are EXPORTED from system files:

```
:hree_bits          : 0..7;
four_bits           : 0..15;
six_bits            : 0..63;
thirteen_bits       : 0..8191;
byte                : 0..255;
two_bytes           : 0..65535;
```

hese values may be used to set aside header fields of various lengths.

```
string_80 = STRING[80];   { There are 80 characters in one screen row }

incomplete_hdr_str = '      ***** Incomplete Header      *****';
```

'his is a string constant that should be written to the display manager if a header display was not omplete.

:ertain modules must be IMPORTED for their procedures and constants to be accessed. These nodules include:

segmenter
sysglobals
iocomasm
rcvrbuffers
lan_lib
config_net
selfrmt
dspmgr

In order to import these modules, a search statement must be included at the beginning of the application source file: $SEARCH'SYS_72'$

Helpful Hints

The programmer may construct the application code to best suit the needs of the decode at hand. However, the structure of the current application code may provide a good framework from which build future decodes. A description of that structure follows:

- The decode is invoked by the calling environment.

- The header bytes are pulled from memory using 'get_data_bytes'. Remember, don't extract more bytes than 'protocol_end' will allow! Clever use of case-variant records can greatly simplify access to fields within the frame. If a static data structure can be defined which describes the protocol fields, a case-varient record can be used to convert the string returned by 'get_data_bytes' into that form. An example would be:

```
PE
:ader_x_fields_type = PACKED RECORD
                        header_length      : byte;  { Always leave this byte for a string length
                        field_1            : two_bytes;
                        field_2            : thirteen_bits;
                        field_3            : three_bits;
                        field_4            : two_bytes;  etc.

                      END;

:ader_x_type = RECORD
                 CASE BOOLEAN OF
                   TRUE  : ( header_x_data : string_xx );
                             Where xx = header length in bytes }
                   FALSE : ( header_x_fields : header_x_fields_type )

               END;

R
header_x   : header_x_type;
```

en get_data_bytes is called, the VAR put_bytes_here (described earlier in this chapter) would ome header_x.header_x_data. The first byte would then contain the string length and each field ıld be superimposed over its proper label. The values could be accessed as der_x.header_x_fields.field_1, etc., thereby making the code more readable. Note: A WITH ement is useful in this case.

- Next, the return parameters are calculated and set. Calculations may involve various fields from the header that was obtained using the get_data_bytes procedure.

- The headers are now ready to be displayed. First, the programmer checks to see if the displa for the current_level is turned on. If not, the decode may return to the calling environment.

- If the display is on, the user checks to see if the variable format.hdrs[current_level].hexbytes_on is TRUE. If it is, the application calls display_hexbyte_hdr with the appropriate number of bytes in the header.

- Then the programmer checks to see which display to use (summary or detailed) and calls the proper routine.

} The Decode Application

- Within the separate display routines, a separate procedure was used to generate each string to be displayed. Once the string was generated, the display would send the string and its attributes to the display manager, and then call a procedure to generate the next string. The process continues until each string has been displayed.

- If a header display is incomplete, and 'incomplete header' message is sent to the display manager.

e User List

ing and Removing Lists

rocedures within the HP 4972A base system for adding a new user list module to the system are ibed here. These procedures allow an application to set up its own user editable lists and connect into the the edit lists, print lists, and disc functions menus. Applications can use these lists to name to address mappings that can be used as part of the decode presentation. An example list IP address list that is part of the TCP/IP decode application.

leclarations from module userlists (file userlists) are given below.

```
list_name_type = string_9;

dure: add_user_list( list_name          : list_name_type;
                     skupperline        : softkey_line;
                     sklowerline        : softkey_line;
                     dir_mnemonic       : filetypes;
                     dir_suffix         : file_suffix;
                     load_proc          : load_procedure_ptr;
                     save_proc          : procedure_pointer;
                     VAR successful     : BOOLEAN

dure: remove_user_list( list_name       : list_name_type;
                        dir_mnemonic    : filetypes );
```

user_list places the list name in a data structure holding the names of lists currently known to the n. The list name will then show up in the Edit Lists and Prints Lists menus. Add_user_list also izes disc functions for handling files containing this type of list functions.

ve_user_list removes the list name from those currently known to the system. The list is ed of, freeing up the memory it occupied. It no longer shows up in the Edit Lists or print Lists s. Disc functions no longer allows Deleting, Saving, Loading, or Copying of this type of list file.

ımeter Descriptions

ıame : The name of the list. Cannot exceed 9 characters. Can contain only alphanumeric characters plus the underscore character '_'. Must be uppercase. The list name is used in the formation of procedure names which the base system can use to access this list. It must match that part of the procedure's name. It must also match the name of the module containing the list editing routines, since this is used in forming the procedure names as well. This is shown in further detail later.

perline, sklowerline : Gives the softkey label that disc functions will use for selecting this type of list file. The softkey will appear in the Delete File, Save File, Load File, and Copy File functions.

mnemonic : Gives the label that will be shown by disc functions during a directory listing for files containing this type of list.

suffix : the one character file extension used to identify files containing this type of list.

d_proc : Tells mass store what procedure to call for loading this type of file. Should always use "load_this_file" for any new file types.

ɜ_proc : Tells disc functions what procedure to call for saving this type of list file. The procedure should be the "save_xxxxxxx" procedure exported from the list module, where "xxxxxxx" is the same as the list_name.

cessful : Returned true if the addition of the list was successful.

ımple calls

```
add_user_list( 'TCPPORT',
               'TCP Port', 'File',
               'TCP Port', 't',
               load_this_file,
               save_tcpport,
               status );

remove_user_list( 'TCPPORT', TCP Port' );
```

User List Module Description

This section describes the basic outline of a HP4972A user list module. A user list module allows application to define its own user editable lists. It provides the routines to create, edit, save, load, print, and dispose of a user editable list. Applications can use these lists to allow name to address mappings that can be used as part of a decode presentation. An example is the IP address list tha part of the TCP/IP decode application.

The interface between the base system and a user list module is described in the previous section.

List Module Required Procedures

Every module for defining and editing a user list must export the procedures listed below. "xxx" is in place of the list name in the table below. Substitute the name of your list in the place of "xxx" (s the list_name parameter description given above).

Procedure Name	Purpose
xxx	Module name
edit_xxx_list	Edit list main procedure
init_xxx_list	List allocation and initialization
dispose_xxx_list	List deallocation routine
get_xxx_name	List access procedure *
get_name_given_xxx	List access procedure *
get_xxx_given_name	List access procedure *
get_xxx	Mass store save entry routine
put_xxx	Mass store restore entry routine
append	Mass store restore entry routine
default_xxx_list	Check for default list routine
print_xxx_list	Print list routine
get_xxx_label	Returns softkey label for edit list menu
Save_xxx	Mass store save routine

* not required, but provides means to access contents of list.

Brief descriptions of these procedures are given below.

Module name:

- The name of the module containing the list, as declared in the module statement.
- The module name must match the list_name parameter used in the call to the add_user_l procedure.
- The module name (as passed to the add_user_list procedure) is used by the base system t form the names of procedures it expects to be exported from the list module.
- To prevent similar procedures within different list modules from having the same name, t module name is included within the names of exported procedures. This part of the proc name must match the module name.
- The module name cannot exceed nine characters.
- Can contain only alphanumeric characters plus the underscore character '_'. The compil treats upper and lower case as the same.

Edit_xxx_list : (xxx is module name)
This is the main procedure for editing the list. It is called from the edit list menu when the user selects to edit this list.

Init_xxx_list
This procedure should be called by the application initialization procedure after a successful call to add_user_list. It should create any data structures needed for the list and do any necessary initialization. Typically, init_xxx_list would allocate the list data structures from heap memory.

Dispose_xxx_list List deallocation routine
This procedure disposes of the list. It frees up any dynamically allocated data structures by returning them to the heap. This procedure would normally be called by the application unload procedure.

Get_xxx_name List access procedure
Allows access to the list. Given the entry number for a list entry, get_xxx_name returns the name for that element.

Get_name_given_xxx — List access procedure

Allows access to the list. Given a field value for a list entry, get_name_given_xxx returns the name for that element. As an example, get_name_given_tcpport returns the name for the passed in port number. The port name returned is then used as the next level protocol name as the label for the port in the TCP display.

Get_xxx_given_name — List access procedure

The corollary of get_name_given_xxx.

Get_xxx — Mass store save entry routine

Called repetitively by the disc function save_generic procedure to save the list in a file. Each call returns a string of bytes representing the requested entry from the list.

Put_xxx — Mass store restore entry routine

The corollary to get_xxx. Put_xxx is called repetitively to put each table entry into a disc file. Put_xxx should reset the table to empty when called with entry_number equal to zero. Entry number one will contain the list name. Of course, you can modify get_xxx and put_xxx to store and retreive whatever string records you like.

Append_xxx — Mass store restore entry routine

Similar to put_xxx. This procedure only appends entries to the end of the table. Each call to append_xxx appends one entry to the table. There will be no call to append entry number zero, which would reset the current table contents.

Default_xxx_list — Check for default list routine

This function should return true if the list contains only its default entries. Default_xxx_list is called by disc functions when a list is to be loaded. If default_xxx_list returns false, disc functions will ask the user whether to append the list file or overwrite the current list contents.

print_xxx_list — Print list routine

Print_xxx_list is called from the userlists module to print the list.

Get_xxx_label — Returns softkey label for edit list menu

This procedure must return the softkey label that will be displayed for this list in the Edit List and Print List menus. When the user selects the softkey with this label, either edit_xxx_list or print_xxx_list will be called.

Example Calls

```
edit_xxx_list;

init_xxx_list( VAR success : BOOLEAN );

dispose_xxx_list;
get_xxx_name(   entry_number                : INTEGER;
                VAR name                    : prot_name_type;
                VAR dont_care_present       : BOOLEAN;
                VAR found                   : BOOLEAN );

get_name_given_xxx(  xxx_value              : xxx_value_type;
                     xxx_dont_care_mask     : xxx_mask_type;
                     VAR name               : prot_name_type;
                     VAR found              : BOOLEAN );

get_xxx_given_name(  name                   : prot_name_type;
                     VAR xxx_value          : xxx_value_type;
                     VAR xxx_dont_care_mask : xxx_mask_type;
                     VAR found              : BOOLEAN );

get_xxx(   entry_number                     : INTEGER;
        VAR entry_str_ptr                   : ptr_str255;
        VAR error                           : BOOLEAN );

put_xxx(    entry_number                    : INTEGER;
        VAR entry_str_ptr                   :  ptr_str255;
        VAR error                           : BOOLEAN );

append_xxx(    entry_number                 : INTEGER;
            VAR entry_str_ptr               : ptr_str255;
            VAR error                       : BOOLEAN );

default_xxx_list( VAR default               : BOOLEAN );

print_xxx_list;

get_xxx_label( VAR sk_label : softkey_label );
```

Save xxx / Var success : boolean /

The Disk Functions Interface

Some procedures within the HP 4972A base system are useful to applications. For mass storage, t
are several. These procedures allow an application to set up its own files and connect them into th
disc functions menus. The declarations are listed below.

```
Type
        filetypes = string[10];
        file_suffix = string[1];

        procedure_pointer  = Procedure( VAR success: BOOLEAN );

        load_procedure_ptr = Procedure(File_name    : file identifier;
                                       VAR success :BOOLEAN );

Procedure add_masstore_file_type( top_line        : softkey_line;
                                  bottom_line     : softkey_line;
                                  dir_mnemonic    : filetypes;
                                  dir_suffix      : file_suffix;
                                  VAR status      : INTEGER );
```

Add_masstore_file_type tells the disc functions software to recognize file names ending in dir_suf
file of type dir_mnemonic. The dir_mnemonic is the description label that shows up when the use
does a directory listing. The top_line and bottom_line parameters give the softkey label that will b
presented by disc functions for this file type during load, save, delete, and copy operations.

When using this routine, watch out for duplicate uses of dir_suffix. Many suffixes are already in u
A,a,C,c,D,d,e,f,i,M,m,N,n,o,P,p,q,s,T,t,X,x.

Example call

```
add_masstore_file_type( 'IP Addr',
                        'File','
                        'IP Address',
                        'i',
                        status );
```

```
Procedure add_load_procedure( dir_nmemonic   : filetypes;
                              procedure_var  : load_procedure_ptr;
                              VAR status     : INTEGER );
```

Add_load_procedure tells the disc functions software which procedure to call to load a file of type dir_mnemonic. The procedure passed should always be load_this_file, as it has been made to han any file type.

Example call

```
add_load_procedure( 'IP Address', load_this_file, status );
```

```
Procedure add_save_procedure( dir_nmemonic   : filetypes;
                              procedure_var  : procedure_pointer;
                              VAR status     : INTEGER );
```

Add_save_procedure tells the disc function software which procedure to call to save a file of type dir_mnemonic. The procedure passed must be exported from the application. Its parameters m match those of the procedure type procedure_pointer given above.

Example call

```
add_save_procedure( 'IP Address', save_ipaddr_list, status );
```

```
Procedure delete_load_save_procedures( dir_nmemonic : filetypes );
```

Delete_load_save_procedures performs the corollary of the add procedures. It removes the file t from the list of active file types so that the file type is no longer presented during load, save, delet and copy operations. The links to the load and save procedure variables are also removed.