CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :		(11) International Publication Number:	WO 97/23076
H04L 12/00	A1	(43) International Publication Date:	26 June 1997 (26.06.97)

(21) International Application Number: PCT/US96/20779

(22) International Filing Date: 13 December 1996 (13.12.96)

(30) Priority Data:
08/575,506 20 December 1995 (20.12.95) US

(71) Applicant: N B NETWORKS [US/US]; 7 Argonaut, Aliso Viejo, CA 92656 (US).

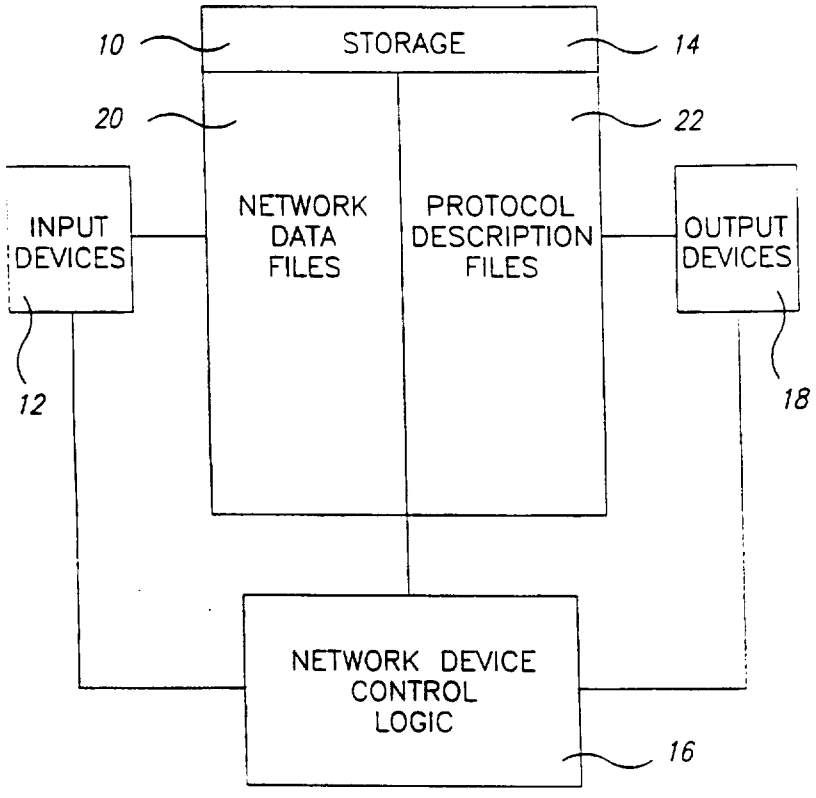
(72) Inventors: BAKER, Peter, D.; 36 Blackbird Lane, Aliso Viejo, CA 92656 (US). NEAL, Karen; 1326 Saltair Avenue #6, Los Angeles, CA 90025 (US).

(74) Agents: BROGAN, James, P. et al.; Lyon & Lyon L.L.P., First Interstate World Center, Suite 4600, 633 West Fifth Street, Los Angeles, CA 90071-2066 (US).

(81) Designated States: AU, CA, CN, IL, JP, MX, SG, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: SYSTEM AND METHOD FOR GENERAL PURPOSE NETWORK ANALYSIS



(57) Abstract

A network interface system and related methods. A single logic control module, which may be implemented in hardware or software, is utilized to perform any of a number of data manipulation functions including, for example, parsing, filtering, data generation or analysis, based upon one or more programmably configurable protocol descriptions which may be stored in and retrieved from an associated memory.

* (Referred to in PCT Gazette No. 44/1997, Section II)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

DESCRIPTION

System and Method for General Purpose Network Analysis

Technical Field

The present invention relates to network communications systems and, in particular, to improved systems and methods for parsing, filtering, generating and analyzing data composed of inter-related structures such as protocols found within network frames.

Background Art

Existing network interface devices provide systems for receiving, analyzing, filtering and transmitting network data or frames of data. Network Protocol Analyzers, Bridges, and Routers are among the most common network interface devices currently available.

Conventional network protocol analyzers provide, for a predefined set of network frame structures or protocols, a system for monitoring the activity of a network and the stations on it by allowing network traffic to be captured and stored for later analysis. Common capture and analysis capabilities include the gathering of statistics, subsequent report generation, the ability to filter frames based on specific criteria, and the ability to generate network traffic.

Bridges and routers are network devices that pass frames from one network interface to another. Bridges operate at the data-link layer and routers at the network layer of the OSI reference model. Like protocol analyzers, both bridges and routers may gather statistics and filter incoming network frames based on specific criteria, however incoming frames also may be forwarded to other networks based on information collected by the bridge or router. Routers typically support only a limited number of network protocols.

Each of these network devices requires an ability to separate network frames into individual protocols and their components (typically referred to as parsing), an ability to filter incoming frames based on a logical combination of one or more field values extracted during parsing, and an ability to gather statistics based in part on extracted field values. Typically, it is a requirement that network frames be received, analyzed and forwarded at full network speeds, sometimes on many different networks at one time.

A frame filter consists of one or more criteria which specify one or more valid values for a frame (or segments of a frame). Frame filtering criteria are typically implemented using an offset (from frame or protocol header start), a length in bits which defines a field, a value for comparison, and mask values for identifying relevant and irrelevant bits within the field. For multiple value filter criteria, the result from each filter value is logically OR'ed together to obtain an overall result. Therefore, each additional result adds to the processing required to filter a given field. For filtering on optional protocol fields that do not occur at the same relative offset in each protocol frame, this method is time-consuming. Thus, it would be desirable to perform filtering on both fixed and optional variable offset fields for any number of values or ranges of values without incurring any additional overhead.

Parsing, the process wherein network frames are broken up into their individual protocols and fields, is necessary for filtering with offsets relative to protocol headers, gathering field based statistics, generating network traffic, routing data frames, verifying field values, and displaying network frames in human readable form. In conventional systems, the parsing process has an overall structure which incorporates control logic for each supported protocol. Therefore, additional control logic must be developed when support for a new protocol is

added to a conventional system. As the development of additional control logic, whether implemented in hardware or software, may be both time consuming and expensive, it would be highly desirable to be able to parse all protocols with a single configurable software (or hardware) module so that support for additional protocols could be added to a system without requiring substantial modification to the system or its control logic.

Further, although microprocessors (or CPUs) available today can execute tens or even hundreds of millions of instructions per second, vendors often must provide dedicated hardware assistance and/or front-end processors with hand-coded assembly language routines to achieve the necessary processing rates for more than one pair of networks. Unfortunately, this solution requires hardware and/or software modifications whenever changes are made to the number of supported features or protocols.

Finally, as networks become larger and more complex, the maintenance of a comprehensive statistics database by each network device becomes more important. Because these statistics databases typically are not utilized by a maintaining device, but instead are collected by a network management device, the collection process may affect performance adversely without any corresponding benefit to the collecting device.

In light of the considerations discussed above, it is believed that a network interface system having a configurable protocol analysis capability with common control logic applicable to many different network devices would be highly desirable.

Summary of Invention

The present invention is directed to improved systems and methods for parsing, filtering, generating and analyzing data (or frames of data) transmitted over a data communications network. In one particularly innovative aspect of the present invention, a single logic control

module, which may be implemented in hardware or software, is utilized to perform any of a number of data manipulation functions (for example, parsing, filtering, data generation or analysis functions) based upon one or more programmably configurable protocol descriptions which may be stored in and retrieved from an associated memory.

The use of common control logic (i.e. the use of a single logic control module) and programmably configurable protocol descriptions allows changes to existing protocols to be made and support for new protocols to be added to a system in accordance with the present invention through configuration only -- without the need for hardware and/or software system modifications. Thus, those skilled in the art will appreciate that a network interface in accordance with the present invention may be configured and reconfigured, if necessary, in a highly efficient and cost effective manner to implement numerous data manipulation functions and to accommodate substantial network modifications (for example, the use of different data transmission hardware, protocols or protocol suites) without necessitating substantial system changes.

In one preferred form, the system of the present invention may employ a CPU or other hardware implementable method for analyzing data from a network in response to selectively programmed parsing, filtering, statistics gathering, and display requests. Moreover, the system of the present invention may be incorporated in a network device, such as a network analyzer, bridge, router, or traffic generator, including a CPU and a plurality of input devices, storage devices, and output devices, wherein frames of network data may be received from an associated network, stored in the storage devices, and processed by the CPU based upon one or more programmably configurable protocol descriptions also stored in the storage devices. The protocol descriptions may take the form of one or more protocol description files for each supported network protocol and may include a protocol

header record and plurality of field sub-records having data corresponding to an associated protocol and fields defined therein.

The system of the present invention also preferably includes logic for extracting field values from particular network frames, performing validation and error checking, and making parsing decisions based upon field values and information in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for filtering a subset of network frames received from the input or storage devices which satisfy a filter criteria based upon information defined in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for filtering network frames which satisfy a plurality of filter criteria which, if desired, may be joined together by Boolean operators.

The system of the present invention also preferably includes logic for analyzing a filter request by breaking the request into its component criteria to determine whether the result from evaluating a particular filter request criteria when combined with results from earlier criteria can be used to filter (i.e. discard) a particular network frame.

The system of the present invention also preferably includes logic for collecting statistics based upon extracted field values satisfying a statistics criteria based upon information defined in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for determining a next protocol description structure required to continue analyzing a network frame.

The system of the present invention also preferably includes logic for determining a frame length and individual protocol header lengths from extracted field values in a network frame.

The system of the present invention also preferably includes logic for making routing decisions based upon information contained in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for determining display formats based on information contained in the programmably configurable protocol descriptions.

The system of the present invention also preferably includes logic for verifying individual field values and making parsing decisions based on the validity of the value.

The system of the present invention also preferably includes logic for constructing and transmitting network frames with varying field contents based on information contained in the programmably configurable protocol descriptions.

The system of the present invention may be employed in any system where it is useful to be able to examine and perform various operations on contiguous bit-fields in data structures, wherein each data structure is composed of predefined fields of one or more contiguous bits. Further, the system of the present invention is particularly efficient where operations must be performed on a subset of included fields.

Those skilled in the art will recognize that the system of the present invention gains a distinct advantage in size and maintainability over conventional network devices by implementing analysis capabilities for multiple known and unknown protocols using common control logic. Furthermore, the system gains a distinct advantage in speed and efficiency over conventional network devices when the control logic is implemented in hardware or a front-end processor, without incurring the penalty of additional hardware and/or software development when protocol definitions change.

Accordingly, it is an object of the present invention to provide an improved system for network analysis wherein the system may determine which protocols and which protocol fields exist in a network frame (also referred herein as parsing) using common control logic combined with configurable protocol descriptions.

It is yet another object of the present invention to provide an improved system for network analysis wherein the control logic may be implemented in hardware as well as software.

It is yet another object of the present invention to provide an improved system for network analysis wherein each supported analysis capability is configurable even when the control logic is implemented in hardware.

It is another object of the present invention to provide an improved system for network analysis wherein the system may determine whether a particular network frame includes a field that satisfies a particular filter criteria based upon information stored in a programmably configurable protocol description.

It is yet another object of the present invention to provide an improved system for network analysis wherein the system may determine if a particular network frame includes a protocol field that satisfies a particular statistics gathering criteria defined in a programmably configurable protocol description.

It is yet another object of the present invention to provide an improved system for network analysis wherein the system may generate network traffic in the form of frames constructed from selected protocol descriptions with the ability to specify a variety of methods for varying individual field values.

It is still another object of the present invention to provide an improved system for network analysis wherein the system may route network frames (determine the appropriate destination interface) that satisfy a particular routing criteria defined in a programmably

configurable protocol description while providing a capability to specify a variety of methods for varying individual field values during the routing process.

It is still another object of the present invention to provide an improved system for network analysis wherein the system may determine if a particular network frame includes a protocol field that contains a value related to either the overall length of the frame or the current protocol header length.

Brief Description Of The Drawings

Fig. 1 is a block diagram of a network interface system in accordance with one form of the present invention.

Fig. 2 is a diagram representing a set of data records of a typical network frame which may be contained in the data files of the network interface system illustrated in Fig. 1.

Fig. 3 is a diagram representing a set of data records of a protocol description in accordance with one form of the present invention.

Fig. 4 is a diagram representing a control record of an Ethernet protocol description which may be utilized in a network interface system in accordance with one form of the present invention.

Fig. 4a is a diagram representing five defined field sub-records of the Ethernet protocol description illustrated in Fig. 4.

Figs. 4b, 4c, and 4d are diagrams representing lookup structures referenced in Fig. 4a fields 0, 2 and 4 respectively.

Fig. 5 is a diagram representing a control record of an imaginary Generic Protocol description which may be utilized in a network interface system in accordance with one form of the present invention.

Fig. 5a is a diagram representing eleven defined field sub-records of the GP description illustrated in Fig. 5.

Figs. 5b, 5c, 5d, and 5e are diagrams representing lookup structures referenced in Fig. 5(a) fields 1, 3, 7 and 8, respectively.

Figs. 6, 6a, and 6b are diagrams representing the control record and field sub-record of a protocol description structure that allows parsing of optional fields of the GP description shown in Figs. 5 - 5e.

Figs. 7, 7a, and 7b are diagrams representing the control record and field sub-records of a protocol description structure that describes the End Of List option of the GP description shown in Figs. 5 - 5e.

Figs. 8, 8a, and 8b are diagrams representing the control record and field sub-records of a protocol description structure that describes the No Operation option of the GP description shown in Figs. 5 - 5e.

Figs. 9, 9a, and 9b are diagrams representing the control record and field records of a protocol description file that describes the Maximum Frame Size option of the GP description shown in Figs. 5 - 5e.

Figs. 10, 10a, 10b, 10c, 10d and 10e are diagrams representing data records of a filter expression control and associated field filter structures.

Fig. 11 is a flow chart illustrating top level frame parsing control logic in accordance with one form of the present invention.

Fig. 12 is a flow chart illustrating protocol parsing control logic in accordance with one form of the present invention.

Fig. 13 is a flow chart of the field parsing control logic in accordance with one form of the present invention.

Fig. 14 is a flow chart representing value verification, error checking, next protocol and branch

determination control logic in accordance with one form of the present invention.

Fig. 15 is a flow chart representing field filtering control logic in accordance with one form of the present invention.

Fig. 16 is a flow chart illustrating field value extraction and varying control logic in accordance with one form of the present invention.

Description of Preferred Embodiments

Referring now to Fig. 1, a network interface system in accordance with one form of the present invention, generally referred to as 10, may be implemented in a network device including input devices 12, data storage devices 14, analysis control logic 16 for facilitating the input, storage, retrieval, and analysis of network frames, and output devices 18 for forwarding frames or displaying or printing the results of analyses. A data storage device 14 may include a data file 20 of network frames having n protocol data records, wherein each data record contains data stored in a plurality of predefined fields. Protocol description files 22 also may be stored in the data storage device 14. The protocol description files 22 may include a protocol control record and n field sub-records, which together may describe a subset of a network protocol and include rules for analyzing that protocol.

The network device control logic 16 is capable of retrieving a subset of network frames from the input devices 12 or data files 20 which satisfy one or more criteria based upon extracted field values and filtering criteria contained in one or more of the protocol description files 22. The network device control logic 16 also includes logic for determining frame and protocol header lengths, gathering statistics, verification and error checking, determining routes, varying values, and formatting output.

A personal computer or conventional network device, such as an IBM PC (or compatible), Apple Macintosh®, or any Unix®, or Zenix® workstation, protocol analyzer, bridge, router, traffic generator, or similar system may be utilized in accordance with the system of the present invention. The data input devices 12 may comprise any of a number of commercially available network interface devices and may include a conventional keyboard or mouse if required. The data storage devices 14 may take the form of any of a number of commercially available data storage options (such as RAM, ROM, EPROM, or various sized fixed disk drives), and the data output devices 18 may comprise any of a number of commercially available user interface devices, such as CRT displays, monitors, network interface devices and/or printers (if required). The analysis control logic 16 may be implemented as a computer program written in any language suitable for systems programming or may be implemented in hardware if better performance is required. In one presently preferred form, the analysis control logic 16 may be implemented via the programming files set forth in the attached Appendix, which is herein incorporated by reference. However, those skilled in the art will appreciate that the analysis control logic 16 might equivalently be implemented in dedicated hardware using, for example, one or more application specific integrated circuits ("ASICs") or one or more field programmable gate arrays ("FPGAs").

The network interface system 10 of the present invention is preferably implemented on a personal computer, workstation or conventional network device having a 32-bit or larger bus and register set, an optional math co-processor, at least one megabyte of available RAM, and for personal computer and workstation applications, a fixed disk having at least 10 megabytes of available storage space. As shown in the attached Appendix, the analysis control logic 16 may be programmed in the C++ language, with abstract data types defined for

statistics gathering, value verification, next protocol determination, filtering, varying values, checksumming and route determination capabilities, and protocol control and field records.

Referring now to Fig. 2, a data file 20 in accordance with one form of the present invention may include a plurality (n) of protocol header data records and optional Data and Pad records. Each protocol record contains data organized into a plurality of predefined fields. Each field comprises a collection of 1 or more contiguous bits and includes a set of valid values for that field. For example, a particular protocol specification might include a 6 bit header length field that limits the protocol header length to values between 20 and 60 inclusive, thereby excluding values less than 20 and values from 61 to 64.

The number of possible contiguous bit fields for a protocol header of length N bits where N is greater than 1 can be expressed by the following formula:

$$\text{Number of Possible Fields} = \sum_{1=1}^{N} i$$

It will be appreciated by those skilled in the art that any possible organization of fields for any possible protocol specification is contemplated for the network interface system 10 of the present invention.

Referring now to Fig. 3, a protocol description file 22 in accordance with one form of the present invention may include a protocol control record, and a plurality (n) of field data records. In a particularly preferred embodiment, the protocol control record (shown below in Table 1) may define the overall structure of a network protocol and reference other information relating to the network protocol.

TABLE 1

PROTOCOL CONTROL RECORD		
Offset	Name	Description
0-3	name_length	length of protocol name in bytes including NULL terminator
4-7	protocol_name	name of protocol control record is describing
8-11	filename	name of file control record is stored in
12-15	numBits	total bit length of protocol header control record is describing
16-17	numFields	number of fields required to describe protocol header
18-19	curField	index of field currently referenced
20-23	outFlag	flag indicating template has been output to file
24-27	dbW	display bit width for protocol header display
28-31	fields	field records that describe protocol header
32-25	options	pointer to option control record to use if this protocol has optional fields
36-39	Rt	pointer to protocol specific routing table

The field records referenced at bytes 28-31 in the table above are preferably organized as shown in Table 2:

TABLE 2

FIELD SUB-RECORDS		
Offset	Name	Description
0-3	fplen	flag indicating value is actual length of frame (multiplier)
4-7	fblen	length of field in bits
8-11	fdwoff	byte offset from start of protocol header of 32-bit field containing value
12	fshl	number of bits to left shift 32-bit value
13	fshr	number of bits to right shift 32-bit value

14	ffmt	number indicating a display type (i.e., decimal, hex, ...)
15	flflag	flag indicating value is actual length of protocol header (multiplier)
16	reserved	not used ... pad byte to align following fields
17	fmult	multiplier to apply to value prior to display
18	fswap	flag indicating the need to swap bytes and words in 32-bit field containing value
19	fsdspfield	flag indicating that this field should be displayed
20-23	fname	pointer to field name (0=none)
24-27	ptr2stats	pointer to configured statistics structure/class (0=none)
28-31	ptr2np	pointer to lookup structure/class ... next protocol definition to use (0=none)
32-35	ptr2vary	pointer to vary field value structure/class (0=none)
36-39	ptr2csum	pointer to checksum structure/class (0=none)
40-43	ptr2flt	pointer to filter criteria structure (0=none)
44-47	ptr2rte	pointer to Route Table structure/class (0=none)

The statistics records referenced in Table 2, above, at bytes 24-27 are preferably organized as shown in Table 3:

TABLE 3

STATISTICS STRUCTURE/CLASS RECORD		
Offset	Name	Description
0-3	StatName	pointer to user assigned name for statistic
4-7	Stat	pointer to derived structure/class for accumulating configured statistic

The next protocol lookup records referenced in the field sub-record table (Table 2) at bytes 28-31 are preferably organized as shown in Table 4:

TABLE 4

LOOKUP STRUCTURE RECORD		
Offset	Name	Description
0-3	Protocol	pointer to protocol description structure
4-7	Next Index	index of field in protocol description to parse next
8-11	Minimum	minimum acceptable value for this range
12-15	Maximum	maximum acceptable value for this range
16-19	okbits	selects even only, odd only, or all values in range
20-23	Translation	pointer to associated human language equivalent

Lookup structures can be used for determining the next protocol control record to use, terminating protocol processing on illegal values, branching decisions for variable length headers or overlapping fields, and for translation of numeric values to mnemonic or written language equivalents. This ability to specify branches on field values allows protocols with multiple overlapping structures to be specified and parsed dynamically.

The vary field value records referenced in the field sub-record table (Table 2) at bytes 32-35 are preferably organized as shown in Table 5:

TABLE 5

VARY FIELD VALUE RECORD		
Offset	Name	Description
0-3	mask	mask for isolating field bits from 32-bit field
4-7	notmask	mask for isolating bits not in field
8-11	operand	value to apply to field bits (relative to field)
12-15	minvalue	minimum allowable value for field bits (relative to field)
16-19	maxvalue	maximum allowable value for field bits (relative to field)

The checksum records referenced in the field sub-record table (Table 2) at bytes 36-39 are preferably organized as shown in Table 6:

TABLE 6

CHECKSUM RECORD		
Offset	Name	Description
0-3	verify	pointer to routine to verify protocol checksum
4-7	compute	pointer to routine to compute protocol checksum

The filter criteria records referenced in the field sub-record table (Table 2) at bytes 40-43 are preferably organized as shown in Table 7:

TABLE 7

FILTER CRITERIA RECORD		
Offset	Name	Description
0-3	Index	index of this filter criteria (zero-based)
4-7	ChPtr	pointer to parent filter channel
8-11	Ranges	pointer to lookup structure containing all possible field values
12-15	Ptl	pointer to associated protocol definition for this criteria
16-19	Fld	pointer to associated field definition for this criteria

The filter channel records referenced in the Filter Criteria record (Table 7) above at 4-7 are preferably organized as shown in Table 8:

TABLE 8

FILTER CHANNEL RECORD		
Offset	Name	Description
0-3	NextCriteriaIndex	index of next criteria that should be applied to this filter
4-7	TotalCriteria	number of criteria required to implement this filter
8-11	Criteria	pointer to array of TotalCriteria criteria structures
12-15	ChannelName	pointer to user supplied filter channel name

Each configured filter consists of one or more filter criteria and the filter criteria may be organized into Filter Criteria records. The Filter Criteria records may refer to lookup structures which allow the filter criteria to determine from a field value the current state of the filter expression at each criteria. These states may include: PASS_FRAME (accept this frame) and FILTER_FRAME (discard this frame).

The NextCriteriaIndex field referenced in Table 8 above at bytes 0-3 is used to ensure that all filter expressions are applied in the required order. The Ptl and Fld fields at bytes 12-19 allow filter criteria to be associated with specific protocols and protocol fields. The lookup records referenced in the Filter Criteria record (Table 7) at bytes 8-11 are preferably organized as shown in Table 9:

TABLE 9

FILTER LOOKUP STRUCTURE RECORD		
Offset	Name	Description
0-3	Return Value	PASS_FRAME, FILTER_FRAME value range result
4-7	Index	index of field in Filter Expression structure
8-11	Minimum	minimum acceptable value for this range
12-15	Maximum	maximum acceptable value for this range

16-19	Mask	selects EVEN, ODD or all values in range
20-23	Translation	pointer to associated human language equivalent

The Route Table records referenced in the Field Sub-Records table (Table 2) at bytes 44-47 are preferably organized as shown in Table 10:

TABLE 10

ROUTE TABLE RECORD		
Offset	Name	Description
0-11	NetMask	mask for extracting 1 to 96 bits from protocol header route field
12-15	entries	number of entries in Route Table
16-19	Table	pointer to array of `entries' Route Table entries

The Route Table Entry records referenced in the table above at bytes 16-19 are preferably organized as shown in Table 11:

TABLE 11

ROUTE TABLE ENTRY RECORD		
Offset	Name	Description
0-11	DstNetAddr	Route Table Lookup Value (field value is compared with this)
12-13	DstFrameType	destination frame type (i.e., 802.3, FDDI, etc.)
14-15	DstInterface	destination interface number
16-17	MacHdrLen	length of MAC header and encapsulation to append to frame
18-19	DataLen	length of frame less MAC header and encapsulation
20-21	MinLen	minimum allowable frame size for this destination
22-23	MaxLen	maximum allowable frame size for this destination

24-27	MacHdr	pointer to MAC header and encapsulation to append to frame
28-31	DataPtr	pointer to frame less any bits below the routing protocol header

In Tables 1-11 the records of the protocol description and associated field, statistics, lookup, checksum, vary, route determination and filter records are shown as they appear when resident in memory. In the presently preferred embodiment, each of these protocol description records with its associated field, statistics, lookup, and filter record information is also written to a protocol specific protocol description file (PDF).

In the presently preferred embodiment, the following sequence of data is written to a PDF:

For the Protocol Control Record:

the length of the protocol name including NULL terminator,
the name of the protocol,
the total bit length of the protocol header,
the number of fields required to describe records,
the index of the field currently referenced,
the display bit width,
for each of the field records that describe the protocol header,
a call is made to write the field related data (This sequence is described below).
if the pointer to the option control record is NULL, zero,
if there are options, the length of the protocol option name
including the NULL terminator,
the option name,
the option's protocol control record

For the Field Record:

the flag indicating the value is the actual length of frame,

the length of the field in bits,

the byte offset from the start of the protocol header,

the number of bits to left shift of the 32-bit field,

the number of bits to right shift of the 32-bit field,

the number indicating the display type,

the flag indicating the value is the actual length of the protocol header,

the reserved byte,

the multiplier to apply to value prior to display,

the flag indicating whether to byte swap the 32-bit value,

the flag indicating that the field is to be displayed,

the length of the field name including the NULL terminator, or zero

if the pointer to the statistics structure/class is NULL, zero

if the pointer to the statistics structure/class is not NULL, a call is made to write the statistics type

if the pointer to the lookup structure/class is NULL, zero

if the pointer to the lookup structure/class is not NULL,

a call is made to write the lookup type, the number of lookups, and the lookup values

The pointer to vary field, pointer to checksum, pointer to filter, and pointer to route determination are handled similarly.

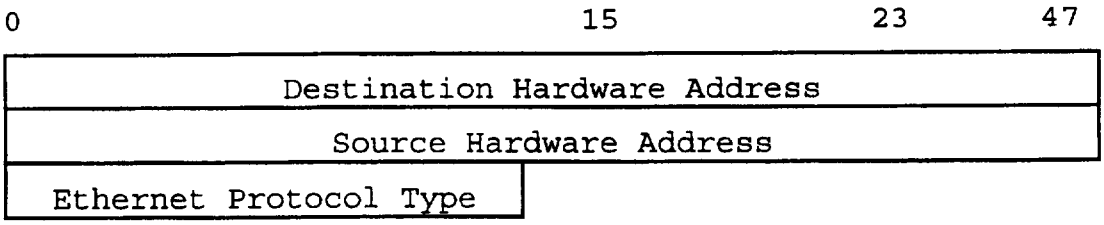
In the presently preferred embodiment, the initialization of the system includes a determination of the

presence of PDF files and the extraction of the protocol and associated control record information from all of the PDF files found. The number of PDF files is determined, and a ProtocolList is constructed consisting of a sorted vector of protocol records at least the size of the number of PDF files found. The name of each protocol record found is inserted in the ProtocolList. The PDF files are then read to memory in the sequence described above for the PDF file writing. The lookup records that indicate a next protocol are associated with the appropriate entries in the ProtocolList.

Two simple protocol descriptions are provided in Tables 12 and 13 (below) to assist in the description of the system of the present invention. The Ethernet Protocol specification shown below is a simplification of an actual Ethernet protocol header.

TABLE 12

ETHERNET v2.0 PROTOCOL SPECIFICATION

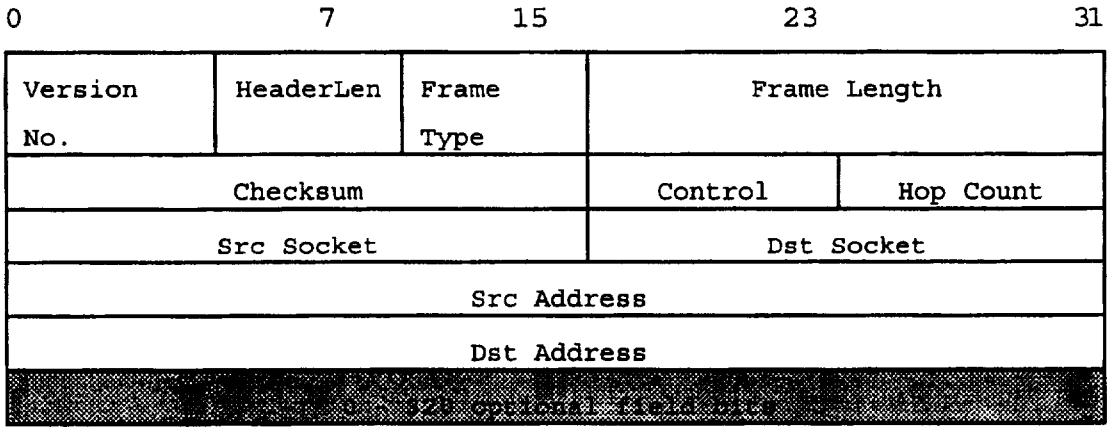


Destination Hardware Address	-	destination hardware station address (48 bits)
Source Hardware Address	-	source hardware station address (48 bits)
Ethernet Protocol Type	-	upper layer protocol designator (16 bits) 0x8888=GP

The Ethernet protocol definition described above specifies only one possible upper level protocol, the Generic Protocol (GP) which is indicated by placing a

hexadecimal 0x8888 value in the protocol type field. The Generic Protocol (GP) specification shown below in Table 13 has been fabricated to provide examples of different types of field functionalities found in actual network protocols.

TABLE 13
GENERIC PROTOCOL (GP) SPECIFICATION



Version Number (4 bits)	-	software version number
HeaderLen (4 bits)	-	length of GP header in 32 bit words. 0- 4 = illegal 5 = No Optional fields, 6-15 = 32-320 bits of options
Frame Type (8 bits)	-	upper level protocol identifier 0 = Unknown 1 = GP1 2 = GP2 3-255 = Unknown
Frame Length (16 bits)	-	frame length in bytes including header
Checksum (16 bits)	-	Checksum of header including options
Control (8 bits)	-	reserved (must be zero)

Hop Count (8 bits)	-	Count of number of networks traversed
Src Socket (16 bits)	-	Socket of Upper-layer protocol sender
Dst Socket (16 bits)	-	Socket of Upper-layer protocol receiver
Src Address (32 bits)	-	Sender protocol address
Dst Address (32 bits)	-	Receiver protocol address

The GP options have two possible formats. Type 1 consists of a single 8-bit option type field containing an option type value. Type 2 contains the 8-bit option type field, but also contains an 8-bit option length field to allow implementation of variable length options in the GP. Two type 1 options and one type 2 option defined in the GP specification are shown below in Table 13:

TABLE 13

End of Option List	(8 bits)	Indicates end of options list. Consists of an 8-bit option type field with value 0. Necessary only for list that does not end on 32-bit boundary.
No Operation	(8 bits)	Performs no function. Consists of an 8-bit option type byte with value 1. Used for alignment of other GP options.
MinMax Size	(3 2 / 4 8 bits)	Allows minimum and maximum allowable frame lengths to be specified. Consists

of an 8-bit option type field with value 2, an 8-bit option length field, an 16-bit minimum frame length field, and an optional 16-bit maximum frame length field. If the maximum frame length is specified, the option length field will have value 4, otherwise it will have value 6 specified in units of 8-bit bytes.

Describing the flow charts of Figs. 11-16 requires the definition of several variables. These variables (described in Table 14 below) are used to implement and monitor the current control logic state of a network interface system in accordance with the present invention:

TABLE 14

FramePtr	-	Pointer to start of network frame being parsed
HwLen	-	Bit length of network frame as reported by input device
ParseLen	-	Number of bits parsed in the current network frame
Current Protocol	-	Pointer to protocol description control record in use
CurField	-	Index of field in CurrentProtocol being parsed
ParsePtr	-	Pointer to start of protocol header in frame being parsed

FrameLen - Number of meaningful bits in the current network frame

ProtoParse Len - Number of bits parsed in the current protocol header

HeaderLen - Size in bits of protocol header being parsed

ParseLvl - Zero based protocol level in ISO reference model of protocol being parsed (current protocol)

ParseList - Array of pointers to protocol headers in frame being parsed (only 0 to (ParseLvl-1) are valid)

SrcIntf - Index of interface on which frame being parsed was received (useful for bridging applications and interface operations)

IntfTypes - Array of values defining the type of each interface in the network system (useful for bridging operations and type specific operations)

Network frames contain one or more protocol headers, an optional number of data bits, and an optional number of pad bits. Frame bits up to the frame length extracted during parsing for which no protocol description exists are considered data. Bits beyond the frame length extracted during parsing are considered to be pad. Two network frames are provided as examples to be used during discussion of the control logic of the present invention:

Frame 1 shown below has a hardware length of eighty-two 8-bit bytes and consists of a fourteen byte Ethernet header, a twenty byte GP header with no option bytes, and forty-eight bytes of application data. No hardware pad is required because the frame length exceeds the Ethernet minimum of sixty bytes.

Frame (1)

(1)	08	00	00	00	00	03	08	00	00	00	00	04	88			
	88													Ethernet Header(14)		
(2)	35	00	00	44	B1	5F	00	01	08	00	01	47	01	02	03	04
	05	06	07	08										GP Header(20)		
(3)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00						Data (24)		
(4)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00						Data (24)		

Frame 2 shown below has a hardware length of sixty 8-bit bytes and consists of a fourteen byte Ethernet header, a twenty-eight byte GP header including eight option bytes, and eighteen bytes of pad to achieve the sixty byte Ethernet minimum frame length requirement.

Frame (2)

(1)	08	00	00	00	00	01	08	00	00	00	00	02	88			
	88													Ethernet Header (14)		
(2)	37	03	00	2A	FF	FF	00	05	08	00	01	00	01	02	03	04
	05	06	07	08										GP Header (20)		
(3)	01													GP NoOp Option (1)		
(4)	02	04	00	00										GP MaxSizeOpt (4)		
(5)	00													GP EOL Options (1)		
(6)	00	00												GP Option Pad (2)		
(7)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	00	00												Frame Pad (18)		

A flow chart is provided for each of the major control logic components of the present invention. The flow chart shown in Fig. 11 outlines ParseFrame control logic in accordance with the present invention and shows how successive protocol headers are parsed, and how remaining information is parsed as application data and frame pad. The flow chart in Fig. 12 outlines ParseProtocol control logic in accordance with one form of the present invention and shows how fixed and optional fields may be parsed in a selected protocol. The flow chart shown in Fig. 13 outlines ParseFields control logic

in accordance with the present invention and shows how decisions are made and operations performed on extracted field values. The flow chart shown in Fig. 14 outlines ValidateValue control logic in accordance with the present invention and shows how branching, next protocol determination, and validity decisions are made with extracted field values. The flow chart shown in Fig. 15 outlines ApplyFilter control logic in accordance with one form of the present invention and shows how filter criteria are applied to fields in network frames. The flow chart shown in Fig. 16 outlines GetValue control logic in accordance with one form of the present invention and shows how field values are extracted from network frames. These six components of the control logic of a network interface system in accordance with the present invention are described in detail below.

Referring now to Fig. 13, a value is extracted by the GetValue control logic (shown in Fig. 16) of the system (at 210) for each configured field that is relevant to the current network frame. As shown in Fig. 16, the fdwoff value is added to ParsePtr to locate and extract a 32-bit value (at 502) containing the field value, which is then byteswapped (at 510) if the fswap field is TRUE. If the ptr2vary field is NULL (at 512 or 518), indicating that the value does not need to be modified or varied, the field value is extracted (at 506) by shifting the 32-bit value left fshl bits and then right by fshr bits to eliminate any extraneous bits. If the ptr2vary field is non-NULL, the extracted 32-bit value is modified using the configured method (at 514 or 520) and the resulting 32-bit value overwrites the extracted 32-bit value (at 516 or 522). If the extracted value was byteswapped (at 510), the modified value is swapped back (at 516) prior to overwriting the original value. The extracted value is returned (at 508) by the GetValue control logic.

It will be appreciated by those skilled in the art that the system of the present invention can handle

different machine endian architectures (at 210) by rearranging the bit and byte order of the extracted 32-bit network frame value to match the target hardware architecture, and can be adapted easily to RISC based architectures where all memory accesses must be aligned in some fashion.

If the GetValue control logic was about to extract a value for the GP HeaderLen field from frame (2), ParsePtr would point at the first value of line 2, from Fig. 5a, fdwoff would be 0, fshl would be 4, and fshr would be 28, so that 32-bits of data would be extracted (at 502) and, possibly, byteswapped (at 510) to obtain a hexadecimal value equal to 0x3703002A.

In binary notation this is:

0011 0111 0000 0011 0000 0000 0010 1010

Shifting left 4 bits (at 506) yields:

0111 0000 0011 0000 0000 0010 1010 0000

Shifting right 28 bits (at 506) yields:

0000 0000 0000 0000 0000 0000 0000 0111

Which in decimal notation is: 7

Therefore, the actual length of the GP header in frame (2) is seven 32-bit words, which is 28 bytes or 224 bits.

Although the presently preferred embodiment of the system of the present invention is designed to handle a maximum field width of 32 bits, it will be appreciated by those skilled in the art that the system may be designed to handle any required maximum field width, and is particularly efficient where the maximum field width matches the underlying hardware architecture. It will also be appreciated by those skilled in the art that it is possible to divide larger protocol fields into sub-fields as demonstrated by the Ethernet protocol field descriptions shown in Fig. 4a where the 48-bit hardware address fields have each been defined as two 24-bit sub-fields.

The ValidateValue control logic shown in Fig. 14 is performed on each extracted field value by the ParseFields control logic (at 214) shown in Fig. 13. Each field may have an associated lookup structure reference containing one or more values and/or ranges of values that have a particular meaning for that field.

If no lookup structure is configured for a particular field, all values are deemed to be valid (at 318 and 312), which causes parsing to continue with the next sequentially defined field of the current protocol description.

If a lookup structure exists for a particular field but the extracted value is not found therein (at 314 and 316), parsing still continues with the next defined field of the current protocol. However, the value is considered invalid.

Values or ranges of values found in configured lookup structures are considered to be valid. The Prot and NextIndex values associated with a value or range found are used to specify NextProtocol, the protocol description (at 308) to be used after current protocol header parsing is completed, and the index of the next field (at 310) is used to determine where parsing of the current protocol will continue after the current field. The first valid field parsed in a protocol that specifies the NextProtocol has precedence over all subsequent NextProtocol specifiers (at 306).

The ValidateValue control logic returns an updated CurField value (at 312 and 316) together with a valid/invalid indication, and where indicated (at 308) may return an updated value for NextProtocol.

Using value 0x8888 as an example, if the ValidateValue control logic is applied to the Ethernet Type field and associated lookup structure shown in Figs. 4a and 4d respectively, the lookup structure would be found (at 302), the value will be found in it (at 304), the associated Protocol field found with the range

containing 0x8888 value will be used to update the NextProtocol variable (at 308) if it is NULL (at 306), and the associated Next Index field will be used to update the CurField variable.

Using Fig. 5c as an example, it may be seen how values may be used to continue parsing at different locations in the current protocol description. In this case, value 0x02 for the Frame Type field causes Checksum field processing to be skipped.

Referring back to the ParseFields control logic shown in Fig. 13, the ParseFields control logic parses the fields in each protocol header contained in a particular network frame by using field values obtained in accordance with information specified in associated protocol descriptions. The ParseFields control logic is applied for each protocol description required for a particular network frame. If the ParseFields control logic were applied to the exemplary frame, "Frame (1)," described above, the network interface system 10 of the present invention would apply the ParseFields control logic with the protocol descriptions for the Ethernet protocol shown in Table 12, the GP shown in Table 13, and an unspecified Data protocol description.

The ParseFields routine is entered (at 200) with ParsePtr pointing at the start of a protocol header in a particular network frame and CurrentProtocol set to an appropriate protocol description. Parsing starts at Protocol bit and field zero when CurField and ProtoParseLen are cleared (at 202), also, HeaderLen is set to the configured protocol control record NumBits value, and LocalProto, the local next protocol return value variable is cleared. Using the Ethernet protocol description shown in Fig. 4 as an example, HeaderLen would be set to 112 bits.

The control loop (at 204 through 224) continues until the last field has been parsed (at 206), all bits in the

header have been parsed (at 208), or all bits in the frame have been parsed (at 209).

For each field a value is retrieved by the system (at 210). If there is a filter criteria for the field it is applied (at 232) by the ApplyFilter control logic. The System Filter Status is set to FILTER_FRAME and NextCriteriaIndex is set to zero for every configured filter channel prior to the start of frame processing and after each frame is processed (at 124 in Fig. 11).

Referring now to the overall system filter channel control structure shown in Fig. 10, and using the filter expression shown below as an example:

if ((the Ethernet Dst Vendor Address is 0x08FFFF AND the Ethernet Dst Station Address is 0x334455) OR (the GP Frame Type is 1 OR the GP Frame Type is 2))
keep this network frame

we can divide the expression into three distinct filter criteria:

(0) if the Ethernet Dst Vendor Address is 0x08FFFF
(1) if the Ethernet Dst Station Address is 0x334455
(2) if the GP Frame Type is 1 OR the GP Frame Type is 2

Fig. 10(a) shows an example Filter channel structure for the expression shown above and refers to the three Filter Criteria Records of Fig. 10(b) that implement the three filter criteria shown above and refer respectively to Figs. 10(c), 10(d) and 10(e) which implement the three criteria as lookup structures.

Referring now to Fig. 15, after the ApplyFilter control logic is entered (at 400), the Index of one of the filter criteria records shown in Fig. 10(b) is computed with NextCriteriaIndex (at 402 and 404) for the associated filter channel shown in Fig. 10(a).

If Index is less than NextCriteriaIndex (at 402) it indicates that this filter criteria does not need to be

evaluated. This may occur because a filter channel has been satisfied and NextCriteriaIndex has been set to TotalCriteria to disable further filter processing.

If Index is greater than NextCriteriaIndex (at 404) this indicates that a filter criteria was skipped in the evaluation of this filter channel which invalidates the filter result. In this case, further filter evaluation is disabled (at 414) by setting NextCriteriaIndex to TotalCriteria and ApplyFilter returns to the caller.

If Index and NextCriteriaIndex are equal, the field value is found (at 406) in the associated lookup table, NextCriteriaIndex is updated with the associated NextIndex value and if the associated return value status is PASS_FRAME, the System Filter Status is updated to PASS_FRAME. In this preferred embodiment, the range of possible values for a field must be fully covered. Similarly, in the preferred embodiment a frame will be completely parsed for statistics gathering.

Criteria (0) cannot be used to determine a PASS/FILTER_FRAME result for the filter expression above because it must be logically AND'ed with criteria (1). This is illustrated in Fig. 10b, where every value results in no change to the status. The logical AND with criteria (1) is implemented using the NextIndex value. If criteria (0) is FALSE then NextIndex is 2 which causes criteria (1) to be skipped, otherwise NextIndex is 1.

Criteria (1) when TRUE can be used to determine that the filter expression is TRUE because it is not evaluated unless criteria (0) is also TRUE, and the filter expression is the result of ((0) and (1)) or (2). If criteria (2) is FALSE then a PASS/FILTER_FRAME result cannot be determined for the filter expression. This is illustrated by Fig. 10c, where the criteria value results in a PASS_FRAME status, and every other value results in no change to the status. The filter expression Count value is reset on completion of frame processing.

Criteria (2) when TRUE can be used to determine that the filter expression is TRUE because it is logically OR'ed with the result of the first two criteria.

It should be noted that the system of the present invention will collect statistics on all fields evaluated regardless of the decision to pass or filter the frame, which may not be acceptable in some instances. It will be appreciated by those skilled in the art that the system of the present invention may be implemented as sequential parsing loops, so that filtering decisions may be made prior to the application of statistics or other field operations.

It will be appreciated by those skilled in the art that the system of the present invention offers significant advantages over traditional filtering methods by allowing filtering criteria to be specified for any subset of bits in any field, by allowing criteria to be applied to every instance of a field that appears more than once in a network frame, and by providing a simple method for easily specifying ranges of values.

Returning again to Fig. 13 after applying a filter criteria, the extracted value is processed by the ValidateValue control logic (at 214), which updates the NextProtocol and CurField variables and returns a valid/invalid value indication. If ValidateValue returns invalid, parsing of the current field stops (at 216) and restarts with the updated CurField value (at 204), otherwise each configured parsing operation is applied based on the extracted value.

The statistics entity of the field sub-record may be used to indicate categories of statistics to be maintained for each protocol header field (at 218 and 236). Details about mechanisms for collecting statistics are not relevant to the present discussion. However, it will be appreciated by those skilled in the art that the addition of various classes of statistics such as field counters, summing of field contents, and arrays of counters/sums

based on field contents may be used in accordance with the present invention. Using Fig. 5a as an example, it would be possible to configure the FrameLength field to accumulate an array of counts for each possible field value. From this array, the distribution of GP frames sizes is immediately available, and the length of all GP frames and each frame size may be computed.

Although checksum verification/computation (at 217 and 235) and route determination capabilities (at 219 and 237) are not described in detail in Fig. 13, those skilled in the art will recognize that a system in accordance with the present invention may be configured easily to implement those capabilities. Further, exemplary software listings for implementing these capabilities are provided in the Appendix which is attached hereto and incorporated herein by reference. Moreover, upon review the listings in the Appendix entitled csum.asm, checksum.hpp, route.cpp and route.hpp, those skilled in the art will appreciate that the ability to configure IP, TCP, UDP and IPX checksum capabilities may readily be incorporated into a system in accordance with the present invention. The same is true for a general purpose 16 bit ones complement checksum capability. Finally, those skilled in the art will appreciate that the system of the present invention may be configured in virtually infinite ways to implement virtually any desired checksum capability or, indeed, any desired data manipulation function.

Although in the preferred form the Verify checksum control logic is integrated into the ParseFields control logic (at 217 and 235), the result is not used because processing of frames with bad checksums is device dependent. For example, frames with bad checksums would be counted and saved by a protocol analyzer, while a routing device would count and discard them.

An ability to route frames based on values contained in fields of up to 96 contiguous bits is also demonstrated in the software listings included in the Appendix, and

those skilled in the art will recognize that the 96 bit limit may be changed easily to allow for proper handling of protocols with route determination fields longer than 96 bits.

Moreover, those skilled in the art with appreciate that the system of the present invention may be augmented to support virtually any field based operations through modification of the ParseFields control logic loop (at 204-224). For example, it is believed that field based encryption and decryption operations may be added to the system of the present invention with minimal effort.

The HeaderLength field of a protocol description sub-record when non-zero is used to indicate that the extracted value of the current field may be used to compute the length of the current protocol header. The extracted value when multiplied with the configured HeaderLength field value yields the length of the protocol header in the current network frame (at 238). The HeaderLength field is configured to be 32 for the FrameLength field of the GP description shown in Fig. 5a. If HeaderLength is used together with the HeaderLen value extracted from frame (2), an actual GP header length of 224 bits (32 * 7) is calculated. Because the fields defined in Fig. 5a add up to only 160 bits, it will then be possible to determine that the (224 - 160) or 64 bits of optional fields exist in frame (2).

For each field with a valid value, the BitLength field is added to ProtoParseLen, the number of bits parsed in the current protocol, and ParseLen, the number of bits parsed in the network frame (at 222).

The FrameLength field of a protocol description sub-record when non-zero is used to indicate that the extracted value of the current field may be used to compute the length of the current network frame. The extracted value when multiplied with the configured FrameLength value yields the number of meaningful bits in the current frame (at 240). The FrameLength field is

configured to be 8 for the FrameLength field of the GP description shown in Fig. 5a. If FrameLength is used together with the FrameLen value extracted from frame (1), an actual frame length of 336 bits is calculated (8 * 42). Because the hardware length of frame (1) is 480 bits (8 * 60 bytes), it is now possible to determine that the last ((480-366) bits) of frame (2) is pad, added to the frame in this case, to achieve the required Ethernet minimum length of (8 * 60 bytes). In a preferred form, the length computed for the frame is verified against the length provided by the hardware, and the minimum of the two values is be used as FrameLen.

If every field in the current protocol has been parsed (at 206), or every bit in the current protocol header has been parsed (at 208), or every bit in the current frame has been parsed (at 209), parsing of the current protocol terminates. If LocalProto is NULL (at 225) when parsing of the current protocol terminates, ParseProtoLen is added to ParsePtr (at 228) so that it points to the start of the next protocol header in the frame. If LocalProto is not NULL (at 225) when parsing of the current protocol terminates and there are unparsed header bits remaining (at 226), ParseLen and ProtoParseLen are adjusted to account for each unparsed header bit (at 227) before adding ProtoParseLen to ParsePtr (at 228). In every case, ParseFields control logic returns LocalProto (at 230).

Referring now to Fig. 11, the ParseFrame control logic of the present invention, network frames are composed of one or more protocol headers which in turn are composed of one or more predefined contiguous bit fields. The ParseFrame control logic systematically parses through each network frame (at 104 to 108 and 128) until all known protocol headers have been parsed. Any remaining frame bits are parsed as application data (at 110, 112 and 130) and/or pad data (at 114, 116 and 132).

Referring now to Fig. 12, ParseProtocol control logic where all fixed and optional protocol fields are parsed is entered (at 200) with ParsePtr and ParseLen initialized from prior processing. All protocol fields that are fixed in location are parsed (at 152). If all bits in the frame are parsed (at 154) after parsing fixed fields, frame parsing is complete and ParseProtocol returns NULL (at 168). If there are more bits in the frame to parse and the current protocol description supports optional fields (at 156) and the current frame contains optional fields (at 160) they are parsed (at 160 to 166) using the current protocol option control protocol description as a starting point (at 158). Once all options are parsed (at 172) ParseProtocol will return NULL (at 168) if all bits in the frame have been parsed or will return LocalProto (at 170) if more bits remain to be parsed.

Referring again to Fig. 11, once the system has received a network frame (at 100), defined by an interface number (SrcIntf), a frame location (FramePtr) and a hardware length (HwLen), the frame is resolved into its protocol and field components using the system of the present invention.

Using the exemplary frame, "Frame (2)," described above as an example, the system (at 102) in Fig. 11 would obtain from the receiving network interface device SrcIntf, the receiving interface number, FramePtr, a pointer to the frame, and HwLen, the hardware frame length. The hardware length of frame (2) is 480 bits. ParseLen, the number of bits in the frame that have been parsed, ParseLvl and CurField, the index of the protocol field being processed are reset to zero, and CurrentProtocol, is set up with the initial protocol description structure of the receiving interface number which for frame (2) is the Ethernet Protocol description defined in Figs. 4 - 4d. FrameLen is set to the value of HwLen, and ParsePtr is set to the value of FramePtr.

Each field in the Ethernet Protocol description as shown in Fig. 4a is parsed (at 106) using the ParseProtocol control logic shown in Fig. 13.

The ParseProtocol control logic updates ProtoParseLen, the number of bits parsed in the CurrentProtocol, HeaderLen, the protocol header size determined during parsing, and returns NextProtocol, a reference to the next applicable protocol description structure to use during parsing. ParseProtocol also updates ParsePtr and ParseLen by adding ProtoParseLen to them. If NextProtocol is NULL, the remaining frame bits will be treated as Data and/or Pad bits.

After the Ethernet protocol fields in frame (2) are parsed (at 106) by the ParseProtocol control logic shown in Fig. 13, HeaderLen, ParseLen and ProtoParseLen will be 112 bits, NextProtocol will refer to the GP shown in Figs. 5-5(e), and ParsePtr will point at the start of line 2 in frame (2). CurrentProtocol will be updated with the NextProtocol value of GP (at 130) and the GP fields in frame (2) are parsed (at 106) by the ParseFields control logic shown in Fig. 13, which will update HeaderLen and ProtoParseLen to be 160 bits, and return NextProtocol as NULL. ParsePtr will point at the start of line 3 in frame (2), and ParseLen will be updated to 272 bits.

Referring now to Fig. 12, if a CurrentProtocol such as GP shown in Figs. 5 - 5e supports optional fields, which is indicated by the Options field of the control record, then any options in the network frame are sequentially parsed (at 160 - 166) using the ParseFields control logic (at 164) until ProtoParseLen, the number of bits parsed in the protocol header is equal to HeaderLen, the protocol header size determined during parsing with the original protocol description (at 152).

Using the exemplary frame, "Frame (2)," described above as an example, after the GP fields are parsed (at 152), HeaderLen will be updated to 224 bits, while ProtoParseLen will be updated to 160 bits, which indicates

the presence of (224 - 160) bits of optional fields (at 160).

Every protocol description with optional fields will have a master protocol option description, and one option protocol description for each supported option. Using the GP protocol control record shown in Fig. 5 as an example of how optional fields might be described, the Options field will refer to a master option control record similar to Fig. 6. The master option control record will contain one field (see Fig. 6a) that does not contribute to the number of bits parsed (BitLength zero) with an associated lookup structure (see Fig. 6b) for each possible option. Using Fig. 6b as an example, each defined option refers to an appropriate protocol option description. The first field of each option description has an associated lookup structure (see Figs. 7b, 8b, and 9b) that refers back to the master option control record. Fig. 9a shows how optional fields with variable length may be handled by computing the frame length.

Referring now to Fig. 13, in a preferred form the Option Type field in a frame is examined twice, once with the master protocol option description and once with the appropriate option protocol description. If an unknown option is encountered (any value between 0x03 and 0xff inclusive for Fig. 6b), ParseLen, ProtoParseLen, and ParsePtr are updated (at 227 in Fig. 13) to skip any remaining options and parsing of the frame continues with the LocalProto protocol description returned (at 230).

Referring again to Fig. 12, and using the GP control record shown in Fig. 4 as an example, the system would determine (at 156) that the CurrentProtocol supports options and (at 158) will update CurrentProtocol to the master option descriptor of Fig. 6. The master option control record has one field shown in Fig. 6a, which is used to select the appropriate option protocol description structure to use. The lookup structure shown in Fig. 6b allows option descriptions to be associated with option

type values extracted from network frames. The system (at 160-166) will parse one option at a time in the current network frame until all options are parsed.

Before each option is parsed, the number of bits parsed using the previous option protocol control record is subtracted from HeaderLen (at 162). The ParseFields control logic is alternately processed (at 164) with the master protocol option control record, and an appropriate option control record. The CurrentProtocol is updated (at 166) with the NextOption value returned by ParseFields, and the loop is re-entered (at 160).

Using the exemplary frame, "Frame (2)," described above as an example with ParsePtr pointing at line 3, and CurrentProtocol pointing at the GP master option description shown in Fig. 6, it may be seen how a NumBits value of zero prevents the master option description from contributing to the number of bits parsed (at 162), and how ParseFields and ValidateValue use the master option description field to select an appropriate GP option description structure from the lookup structure of Fig. 6b. For frame (2), the first option byte at line 3 contains value 1, which causes the GP NoOp option description structure shown in Fig. 8 to be selected (at 166). The NoOp NumBits value of 8 is added to ProtoParseLen (at 162), and the single field defined in Fig. 8a is parsed at 164. In a preferred form, each option description structure must have a field with an associated lookup structure that is always processed and refers back to the master option description structure.

Thus, for "frame (2)" the option processing control loop (at 160 through 166) is alternately applied with the description structures of Fig. 6 and Figs. 8, 9, and 7. The GP End Of List Option does not refer back to the master option description because it indicates an end to option processing. Any remaining option bits are assumed to be pad and are disregarded so that the check for more

options (at 160 in Fig. 12) will fail and return to frame protocol parsing (at 108 in Fig. 11).

Once all options have been parsed in frame (2) or the system (at 160) determines that the current frame has no optional fields as in frame (1), the system control logic (at 168 or 170) will return to the main parsing loop (at 108 in Fig. 11).

While the invention of this application is susceptible to various modifications and alternative forms, specific examples thereof have been shown in the drawings and are herein described in detail. It should be understood, however, that the invention is not to be limited to the particular forms or methods disclosed, but to the contrary, the invention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the appended claims.

Appendix

The following software was developed using the Watcom C/C++ 10.0 Multi-Platform 16- and 32-bit Development System. The following files are included:

csum.asm
data.cpp
decode.cpp
filter.cpp
lookup.cpp
main.cpp
nxtptl.cpp
pcols.cpp
popfile.cpp
stat.cpp
xmit.cpp
checksum.hpp
decode.hpp
filter.hpp
gen.hpp
lookup.hpp
pa.hpp
pcols.hpp
route.hpp
stat.hpp
xmit.hpp

```
;////////////////////////////////////////////////////////////////////////
;/  CSUM.ASM
;////////////////////////////////////////////////////////////////////////

.586

_TEXT   segment dword public 'CODE'
_TEXT   ends

_DATA   segment word public 'DATA'
;
; Mask values for extracting 1, 2, or 3 misaligning bytes at the front of the
; requested checksum calculation so that the remaining quadword accesses will
; be quadword aligned
;
PreMask:DD   000000000H   ; Address & 3 = = 0 is quadword aligned(not used)
        DD   0ffffff00H   ; Address & 3 = = 1 Mask 3 bytes to next quadword
        DD   0ffff0000H   ; Address & 3 = = 2 Mask 3 bytes to next quadword
        DD   0ff000000H   ; Address & 3 = = 3 Mask 3 bytes to next quadword
;
; Table to convert low 2 bits of starting address into number of bytes of
; quadword misalignment
;
Adjust: DD   0          ; Address & 3 = = 0 is quadword aligned(not used)
        DD   3          ; Address & 3 = = 1 has 3 bytes to next quadword
        DD   2          ; Address & 3 = = 2 has 2 bytes to next quadword
        DD   1          ; Address & 3 = = 3 has 2 bytes to next quadword
;
; Table to extract 1, 2, or 3 bytes from non-quadword multiple checksum lengths
;
Maskit: DD   0          ; Length is a multiple of 4 (not used)
        DD   000000ffH    ; Length is a multiple of 4 plus 1
        DD   0000ffffH    ; Length is a multiple of 4 plus 2
        DD   00ffffffH    ; Length is a multiple of 4 plus 3
;
; Jump Table for entering checksum loop with number not a multiple of 256 bytes
;
```

```
Index: DD   L00      ; Used initial time only for 0 length checksum
       DD   L01      ; 1 quadword to checksum
       DD   L02      ; 2 quadwords to checksum
       DD   L03
       DD   L04
       DD   L05
       DD   L06
       DD   L07
       DD   L08
       DD   L09
       DD   L10
       DD   L11
       DD   L12
       DD   L13
       DD   L14
       DD   L15
       DD   L16
       DD   L17
       DD   L18
       DD   L19
       DD   L20
       DD   L21
       DD   L22
       DD   L23
       DD   L24
       DD   L25
       DD   L26
       DD   L27
       DD   L28
       DD   L29
       DD   L30
       DD   L31
       DD   L32
       DD   L33
       DD   L34
       DD   L35
       DD   L36
```

```
        DD    L37
        DD    L38
        DD    L39
        DD    L40
        DD    L41
        DD    L42
        DD    L43
        DD    L44
        DD    L45
        DD    L46
        DD    L47
        DD    L48
        DD    L49
        DD    L50
        DD    L51
        DD    L52
        DD    L53
        DD    L54
        DD    L55
        DD    L56
        DD    L57
        DD    L58
        DD    L59
        DD    L60
        DD    L61
        DD    L62       ; 62 quadwords to checksum
        DD    L63       ; 63 quadwords to checksum

    _DATA  ends

    DGROUP      group  _DATA
    _TEXT       segment dword public 'CODE'
                assume CS:_TEXT
                assume DS:DGROUP
    ;
    ;   unsigned short csumPseudo(U32 ChkSum, U32 *IpHdr)
    ;         parameters - Sum of IP data (i.e. TCP/UDP header and data)
```

```
;                - pointer to start of associated IP header
;
;          returns   - 16 bit ones complement of the ones complement sum
;                    of the IP pseudo header, and IP data
;
    public csumPseudo_
csumPseudo_ proc   near

    push   ecx
    push   ebx
    mov    cx,+08H[edx]   ; extract TTL and Protocol fields
    mov    ebx,+00H[edx]  ; extract Version, HdLen, TOS, TotalLen fields
    and    ecx,0ff00H     ; isolate Protocol field (remove TTL)
    shl    bx,+0cH        ; isolate HdLen field (remove Version, TOS)
    shr    bx,+02H        ; convert HdLen to byte count in bh
    not    bx             ; invert Hdlen bits to subtract from TotalLen
    add    eax,ebx        ; add TotalLen, subtract HdLen fields
    adc    eax,ecx        ; add Protocol field with carry
    adc    eax,+0cH[edx]  ; add Source Address with carry
    adc    eax,+10H[edx]  ; add Destination Address with carry
    adc    ax,0           ; add final carry
    mov    ebx,eax
    shr    eax,16
    add    ax,bx          ; add upper and lower 16 bit values together
    adc    ax,0           ; add any resulting carry
    not    ax             ; return ones complement of sum
    pop    ebx
    pop    ecx
    ret

csumPseudo_ endp

;
;   unsigned short csumBytes(unsigned long *pktptr, unsigned long iplen);
;          parameter - pointer to associated IP header
;                    (should be passed in EAX)
;          returns   - 16 bit ones complement sum of specified bytes
```

```
;
    public csumBytes_
csumBytes_ proc  near

    push  ebx
    push  ecx
    push  esi
    push  edi
    push  ebp
    mov   ebx,eax          ; ebx = pktptr
    mov   ebp,eax
    xor   eax,eax          ; sum = 0;
    and   ebp,3            ; Is data aligned on DWORD ?
    je    Aligned
    and   ebx,-4H          ; go to next lowest DWORD
    mov   eax,[ebx]        ; get previous quadword
    mov   ecx,PreMask[ebp*4] ; extraction Mask for misaligned bytes
    sub   edx,Adjust[ebp*4]  ; subtract misaligning bytes from length
    and   eax,ecx          ; Mask out unnecessary bytes
    add   ebx,4            ; go to next DWORD for regular processing
Aligned:push  ebp          ; save number of bits to roll result left
    mov   esi,edx          ; length = edx
    shr   esi,02H          ; dlen = esi = length/4
Lswitch:xor   ebp,ebp      ; Clear initial register for checksum
    cmp   esi,00000040H    ; if (dlen < 64)
    jae   L64              ;    (switch default case)
    xor   ecx,ecx          ; Clear Checksum Registers...
    xor   edi,edi          ; and...
    lea   ebx,[ebx+esi*4]  ;    pkt += (dlen - 64)
    sub   ebx,+100H
    jmp   dword ptr Index[esi*4] ; switch(dlen)
L64:  mov   ecx,[ebx]      ; sum += *(pkt+0)
    adc   eax,ebp
L63:  mov   edi,+4H[ebx]   ; sum += *(pkt+1)
    adc   eax,ecx
L62:  mov   ebp,+8H[ebx]   ; sum += *(pkt+2)
    adc   eax,edi
```

```
L61:  mov   ecx,+0cH[ebx]      ; sum += *(pkt+3)
      adc   eax,ebp
L60:  mov   edi,+10H[ebx]      ; sum += *(pkt+4)
      adc   eax,ecx
L59:  mov   ebp,+14H[ebx]      ; sum += *(pkt+5)
      adc   eax,edi
L58:  mov   ecx,+18H[ebx]      ; sum += *(pkt+6)
      adc   eax,ebp
L57:  mov   edi,+1cH[ebx]      ; sum += *(pkt+7)
      adc   eax,ecx
L56:  mov   ebp,+20H[ebx]      ; sum += *(pkt+8)
      adc   eax,edi
L55:  mov   ecx,+24H[ebx]      ; sum += *(pkt+9)
      adc   eax,ebp
L54:  mov   edi,+28H[ebx]      ; sum += *(pkt+10)
      adc   eax,ecx
L53:  mov   ebp,+2cH[ebx]      ; sum += *(pkt+11)
      adc   eax,edi
L52:  mov   ecx,+30H[ebx]      ; sum += *(pkt+12)
      adc   eax,ebp
L51:  mov   edi,+34H[ebx]      ; sum += *(pkt+13)
      adc   eax,ecx
L50:  mov   ebp,+38H[ebx]      ; sum += *(pkt+14)
      adc   eax,edi
L49:  mov   ecx,+3cH[ebx]      ; sum += *(pkt+15)
      adc   eax,ebp
L48:  mov   edi,+40H[ebx]      ; sum += *(pkt+16)
      adc   eax,ecx
L47:  mov   ebp,+44H[ebx]      ; sum += *(pkt+17)
      adc   eax,edi
L46:  mov   ecx,+48H[ebx]      ; sum += *(pkt+18)
      adc   eax,ebp
L45:  mov   edi,+4cH[ebx]      ; sum += *(pkt+19)
      adc   eax,ecx
L44:  mov   ebp,+50H[ebx]      ; sum += *(pkt+20)
      adc   eax,edi
L43:  mov   ecx,+54H[ebx]      ; sum += *(pkt+21)
```

```
        adc    eax,ebp
    L42: mov    edi,+58H[ebx]      ; sum += *(pkt+22)
        adc    eax,ecx
    L41: mov    ebp,+5cH[ebx]      ; sum += *(pkt+23)
        adc    eax,edi
    L40: mov    ecx,+60H[ebx]      ; sum += *(pkt+24)
        adc    eax,ebp
    L39: mov    edi,+64H[ebx]      ; sum += *(pkt+25)
        adc    eax,ecx
    L38: mov    ebp,+68H[ebx]      ; sum += *(pkt+26)
        adc    eax,edi
    L37: mov    ecx,+6cH[ebx]      ; sum += *(pkt+27)
        adc    eax,ebp
    L36: mov    edi,+70H[ebx]      ; sum += *(pkt+28)
        adc    eax,ecx
    L35: mov    ebp,+74H[ebx]      ; sum += *(pkt+29)
        adc    eax,edi
    L34: mov    ecx,+78H[ebx]      ; sum += *(pkt+30)
        adc    eax,ebp
    L33: mov    edi,+7cH[ebx]      ; sum += *(pkt+31)
        adc    eax,ecx
    L32: mov    ebp,+80H[ebx]      ; sum += *(pkt+32)
        adc    eax,edi
    L31: mov    ecx,+84H[ebx]      ; sum += *(pkt+33)
        adc    eax,ebp
    L30: mov    edi,+88H[ebx]      ; sum += *(pkt+34)
        adc    eax,ecx
    L29: mov    ebp,+8cH[ebx]      ; sum += *(pkt+35)
        adc    eax,edi
    L28: mov    ecx,+90H[ebx]      ; sum += *(pkt+36)
        adc    eax,ebp
    L27: mov    edi,+94H[ebx]      ; sum += *(pkt+37)
        adc    eax,ecx
    L26: mov    ebp,+98H[ebx]      ; sum += *(pkt+38)
        adc    eax,edi
    L25: mov    ecx,+9cH[ebx]      ; sum += *(pkt+39)
        adc    eax,ebp
```

```
L24:  mov   edi,+0a0H[ebx]      ; sum += *(pkt+40)
      adc   eax,ecx
L23:  mov   ebp,+0a4H[ebx]      ; sum += *(pkt+41)
      adc   eax,edi
L22:  mov   ecx,+0a8H[ebx]      ; sum += *(pkt+42)
      adc   eax,ebp
L21:  mov   edi,+0acH[ebx]      ; sum += *(pkt+43)
      adc   eax,ecx
L20:  mov   ebp,+0b0H[ebx]      ; sum += *(pkt+44)
      adc   eax,edi
L19:  mov   ecx,+0b4H[ebx]      ; sum += *(pkt+45)
      adc   eax,ebp
L18:  mov   edi,+0b8H[ebx]      ; sum += *(pkt+46)
      adc   eax,ecx
L17:  mov   ebp,+0bcH[ebx]      ; sum += *(pkt+47)
      adc   eax,edi
L16:  mov   ecx,+0c0H[ebx]      ; sum += *(pkt+48)
      adc   eax,ebp
L15:  mov   edi,+0c4H[ebx]      ; sum += *(pkt+49)
      adc   eax,ecx
L14:  mov   ebp,+0c8H[ebx]      ; sum += *(pkt+50)
      adc   eax,edi
L13:  mov   ecx,+0ccH[ebx]      ; sum += *(pkt+51)
      adc   eax,ebp
L12:  mov   edi,+0d0H[ebx]      ; sum += *(pkt+52)
      adc   eax,ecx
L11:  mov   ebp,+0d4H[ebx]      ; sum += *(pkt+53)
      adc   eax,edi
L10:  mov   ecx,+0d8H[ebx]      ; sum += *(pkt+54)
      adc   eax,ebp
L09:  mov   edi,+0dcH[ebx]      ; sum += *(pkt+55)
      adc   eax,ecx
L08:  mov   ebp,+0e0H[ebx]      ; sum += *(pkt+56)
      adc   eax,edi
L07:  mov   ecx,+0e4H[ebx]      ; sum += *(pkt+57)
      adc   eax,ebp
L06:  mov   edi,+0e8H[ebx]      ; sum += *(pkt+58)
```

```
        adc    eax,ecx
    L05: mov   ebp,+0ecH[ebx]      ; sum += *(pkt+59)
        adc    eax,edi
    L04: mov   ecx,+0f0H[ebx]      ; sum += *(pkt+60)
        adc    eax,ebp
    L03: mov   edi,+0f4H[ebx]      ; sum += *(pkt+61)
        adc    eax,ecx
    L02: mov   ebp,+0f8H[ebx]      ; sum += *(pkt+62)
        adc    eax,edi
    L01: mov   ecx,+0fcH[ebx]      ; sum += *(pkt+63)
        adc    eax,ebp
        adc    eax,ecx
        adc    eax,0
    L00: add   ebx,+100H           ; pktptr += 256 bytes
        sub    esi,+40H            ; dlen -= 64 quadwords
        jg     Lswitch             ; if more quadwords ... go checksum them
        and    edx,+3H             ; len &= 3
        je     Finish              ; if len is multiple of quadwords
        mov    edi,[ebx]           ; get following quadword
        and    edi,Maskit[edx*4]   ; *pktptr &= Maskit[len&3]
        add    eax,edi             ; add leftover bytes to checksum
        adc    eax,0
    Finish: pop    ecx            ; retrieve number of misalignment bytes
        shl    ecx,3               ; convert to bits
        jz     NoRoll              ; if was not misaligned
        ror    eax,cl              ; roll checksum by misalignment bits
    NoRoll: mov    ecx,eax
        shr    eax,16
        add    ax,cx               ; add upper and lower 16 bit values
        adc    ax,0                ; return(sum) (not ones complement)
    Lret: pop    ebp
        pop    edi
        pop    esi
        pop    ecx
        pop    ebx
        ret
```

```
csumBytes_ endp

_TEXT  ends
    end

////////////////////////////////////////////////////////////////////////
//    DATA.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"

U32 ParseLevel;
WCValVector<S8 *> ParseList(20);
S8 *FramePtr, *ParsePtr;
U32 HwLen, FrameLen, ParseLen, ProtoParseLen, HeaderLen;
U32 SrcIntf=0, IntfTypes[256] = {ETHERNET, FDDI, TOKEN_RING};
RouteTableEntry *RtePtr;
Filters CfgFilters;            // Configured Filters Object
//
//    Lookup table for bit-swapping a byte value
//
//
S8 bs[256] =     // Table to perform bit-swapping
{
0x00,0x80,0x40,0xc0,0x20,0xa0,0x60,0xe0,0x10,0x90,0x50,0xd0,0x30,0xb0,0x70,0xf0,
0x08,0x88,0x48,0xc8,0x28,0xa8,0x68,0xe8,0x18,0x98,0x58,0xd8,0x38,0xb8,0x78,0xf8,
0x04,0x84,0x44,0xc4,0x24,0xa4,0x64,0xe4,0x14,0x94,0x54,0xd4,0x34,0xb4,0x74,0xf4,
0x0c,0x8c,0x4c,0xcc,0x2c,0xac,0x6c,0xec,0x1c,0x9c,0x5c,0xdc,0x3c,0xbc,0x7c,0xfc,
0x02,0x82,0x42,0xc2,0x22,0xa2,0x62,0xe2,0x12,0x92,0x52,0xd2,0x32,0xb2,0x72,0xf2,
0x0a,0x8a,0x4a,0xca,0x2a,0xaa,0x6a,0xea,0x1a,0x9a,0x5a,0xda,0x3a,0xba,0x7a,0xfa,
0x06,0x86,0x46,0xc6,0x26,0xa6,0x66,0xe6,0x16,0x96,0x56,0xd6,0x36,0xb6,0x76,0xf6,
0x0e,0x8e,0x4e,0xce,0x2e,0xae,0x6e,0xee,0x1e,0x9e,0x5e,0xde,0x3e,0xbe,0x7e,0xfe,
0x01,0x81,0x41,0xc1,0x21,0xa1,0x61,0xe1,0x11,0x91,0x51,0xd1,0x31,0xb1,0x71,0xf1,
0x09,0x89,0x49,0xc9,0x29,0xa9,0x69,0xe9,0x19,0x99,0x59,0xd9,0x39,0xb9,0x79,0xf9,
0x05,0x85,0x45,0xc5,0x25,0xa5,0x65,0xe5,0x15,0x95,0x55,0xd5,0x35,0xb5,0x75,0xf5,
0x0d,0x8d,0x4d,0xcd,0x2d,0xad,0x6d,0xed,0x1d,0x9d,0x5d,0xdd,0x3d,0xbd,0x7d,0xfd,
0x03,0x83,0x43,0xc3,0x23,0xa3,0x63,0xe3,0x13,0x93,0x53,0xd3,0x33,0xb3,0x73,0xf3,
```

```
0x0b,0x8b,0x4b,0xcb,0x2b,0xab,0x6b,0xeb,0x1b,0x9b,0x5b,0xdb,0x3b,0xbb,0x7b,0xfb,
0x07,0x87,0x47,0xc7,0x27,0xa7,0x67,0xe7,0x17,0x97,0x57,0xd7,0x37,0xb7,0x77,0xf7,
0x0f,0x8f,0x4f,0xcf,0x2f,0xaf,0x6f,0xef,0x1f,0x9f,0x5f,0xdf,0x3f,0xbf,0x7f,0xff
};

////////////////////////////////////////////////////////////////////////
//    DECODE.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"
extern HDC          hdc ;
int datalines=0;
int padlines=0;
void protocol::add_field_after(U32 idx, protocol *ptl)
{
field *tmpfs = fs;
++num_fields;
fs = new field[num_fields];
field f((U16)tmpfs[idx].offset(), (U8)tmpfs[idx].shlbits(), (U8)tmpfs[idx].shrbits());
numbits(numbits()+ f.bitlen());

for (U32 i=0; i<=idx; i++)
  {
  fs[i] = tmpfs[i];
  if  (fs[i].lookptr())
    fs[i].lookptr()->inc_index(idx);
  }
fs[idx+1] = f;

if  (fs[idx+1].lookptr()) fs[idx+1].lookptr()->inc_index(idx);
U32 actualoff, off = fs[idx+1].bitoffset() + f.bitlen();
actualoff = off & (~31);
fs[idx+1].offset((U16)(actualoff/8));
fs[idx+1].shlbits((U8)(off - actualoff));
fs[idx+1].shrbits((U8)(32 - f.bitlen()));
if (num_fields > (idx+2))
  {
```

```
    for (i=idx+2; i<num_fields; i++)
      {
      fs[i] = tmpfs[i-1];
      if (fs[i].lookptr())
        fs[i].lookptr()->inc_index(idx);
      off = fs[i].bitoffset() + f.bitlen();
      actualoff = off & (~31);
      fs[i].offset((U16)(actualoff/8));
      fs[i].shlbits((U8)(off - actualoff));
      fs[i].shrbits((U8)(32 - fs[i].bitlen()));
      }
    }
  if (num_fields <= 2) delete tmpfs; else delete[]tmpfs;
  ptl=ptl;                                  // for the compiler
  }

  //
  // Add a protocol field via ptl dialog box insert: Before key
  //
  void protocol::add_field(U32 idx, protocol *ptl)
  {
  field *tmpfs = fs;
  ++num_fields;
  fs = new field[num_fields];
  field f(tmpfs[idx].offset(), (U8)tmpfs[idx].shlbits(), (U8)tmpfs[idx].shrbits());
  numbits(ptl->numbits()+ f.bitlen());
  for (U32 i=0; i<idx; i++)
    {
    fs[i] = tmpfs[i];
    if (fs[i].lookptr())
      fs[i].lookptr()->inc_index(idx);
    }
  fs[idx] = f;
  if (fs[i].lookptr())
    fs[i].lookptr()->inc_index(idx);
  for (i=idx+1; i<num_fields; i++)
    {
```

```
    fs[i] = tmpfs[i-1];
    if (fs[i].lookptr())
      fs[i].lookptr()->inc_index(idx);
    U32 actualoff, off = fs[i].bitoffset() + f.bitlen();
    actualoff = off & (~31);
    fs[i].offset((U16)(actualoff/8));
    fs[i].shlbits((U8)(off - actualoff));
    fs[i].shrbits((U8)(32 - fs[i].bitlen()));
    }

  if (num_fields <= 2) delete tmpfs; else delete[]tmpfs;
  ptl=ptl;
  }

  //
  // Delete a protocol field via nxtptl dialog box delete key
  //
  void protocol::delete_field(U32 idx, protocol *ptl)
  {
  field *tmpfs = fs;
  --num_fields;
  fs = new field[num_fields];
  numbits(numbits() - tmpfs[idx].bitlen());
  U32 tmpfsidxbitlen = tmpfs[idx].bitlen();
  U32 noadjust = 0;
  for (U32 i=0; i<num_fields; i++)
    {
    if (i!=idx)
      {
      if ( (tmpfs[i].bitlen() == tmpfs[idx].bitlen()) &&
           (tmpfs[i].offset() == tmpfs[idx].offset()) &&
           (tmpfs[i].shlbits() == tmpfs[idx].shlbits()) &&
           (tmpfs[i].shrbits() == tmpfs[idx].shrbits()) )
           {
           noadjust = TRUE;
           break;
           }
```

```
        }
      }
    // ON DEC_INDEX: IF LAST FIELD IS BEING DELETED, POINT TO WHAT?
    for (i=0; (i<idx && i<num_fields) ; i++)
      {
      if (tmpfs[i].lookptr())
        tmpfs[i].lookptr()->dec_index(idx);
      fs[i] = tmpfs[i];
      }

    if (idx !=num_fields)
      for (i=idx; i<num_fields; i++)
        {
        fs[i] = tmpfs[i+1];
        if (fs[i].lookptr())
          fs[i].lookptr()->dec_index(idx);
        if (!noadjust)
          {
          U32 actualoff, off = tmpfs[i+1].bitoffset() - tmpfsidxbitlen;
          actualoff = off & (~31);
          fs[i].offset((U16)(actualoff/8));
          fs[i].shlbits((U8)(off - actualoff));
          fs[i].shrbits((U8)(32 - fs[i].bitlen()));
          }
        }
    if (num_fields <= 2) delete tmpfs; else delete[]tmpfs;
    cur_field = U16(idx==0 ? 0 : idx-1);
    // NEED to free memory for the one that's deleted
    ptl=ptl;
    }

    /*
     * Get protocol information from file
     */
    void protocol::get_from_file(FILE *fp)
    {
    // What to do about file name ???
```

```
fread(&num_bits,  sizeof(num_bits),  1, fp); // Read fixed header length in bits
fread(&num_fields, sizeof(num_fields), 1, fp); // Read number of fields in protocol description
fread(&cur_field,  sizeof(cur_field),  1, fp); // Read index of last field displayed during configuration
cur_field = min(cur_field, (U16)(num_fields-1));
fread(&dbW,      sizeof(dbW),      1, fp); // Read value of last requested display width in bits
out_flag = 0;
fs = new field[num_fields];
for (U32 i=0; i<num_fields; i++)
  fs[i].get_from_file(fp);
if (dbW < 8) dbW = 8;
if (dbW > 112) dbW = 112;          // TMP TMP TMP TMP TMP !!!!!!!!!!!!!!!!!!!!!! !!!!!!!!!!!!!!!
//
//   Setup option pointer if any configured
//
U32 tmp;
fread(&tmp,      sizeof(tmp),      1, fp);
if (tmp != 0)
  {
  char *tname = new char[tmp];
  fread(tname,      tmp,          1, fp);
  protocols p(tname, 0), t;  // Reads name_length and name
  ProtocolList->find(p, t);
  opts = t.prot();
  }
else
  opts = 0;
}

    S32 xL, xR, yT=0, yB;
static S32 bitWidth, rowHeight;
extern S32 cxClient, cyClient, cxChar, cyChar, nVscrollPos, nPaintBeg, nPaintEnd;
void protocol::setupOutlineCoords()
  {
  S32 protHeight, protWidth, numRows;

  protWidth  = (cxClient/dbW - 1)*dbW;      // Width in pixels of protocol header
  bitWidth   = protWidth/dbW;               // Width of 1 bit of header in pixels
```

```
    protWidth  = bitWidth*dbW;              // make protWidth an even multiple of bits
    numRows    = (num_bits + dbW - 1)/dbW;    // Number of rows in protocol header + 1 for Title
    rowHeight  = cyChar*2 + 1;              // Height of 1 header row in pixels
    protHeight = numRows*rowHeight;         // Height in pixels of protocol header
    xL         = (cxClient - protWidth)/2;  // Leftmost position of protocol header
    xR         = xL + protWidth + 1;        // Rightost position of protocol header
    yT         = nPaintBeg + rowHeight/2;   // Topmost position of protocol header
    yB         = yT + protHeight + 1;       // Lowermost position of protocol header
    }

/*************************************************************************
****/
    S32 AxL, AxR, AyT=0, AyB;
static S32 AbitWidth;
void protocol::setupOutlineCoordsMult(int continu)
    {
    S32 protHeight, protWidth, numRows;
    char buffer[4];
//  protWidth  = (cxClient/dbW - 1)*dbW;    // Width in pixels of protocol header
    protWidth  = 3*(cxClient/dbW - 1)*dbW/4;    // Width in pixels of protocol header
    AbitWidth  = protWidth/dbW;             // Width of 1 bit of header in pixels
    protWidth  = AbitWidth*dbW;             // make protWidth an even multiple of bits
    numRows    = (num_bits + dbW - 1)/dbW;    // Number of rows in protocol header + 1 for Title
    rowHeight  = cyChar*2 + 1;              // Height of 1 header row in pixels
    protHeight = numRows*rowHeight;         // Height in pixels of protocol header
    AxL        = (cxClient - protWidth)/2;  // Leftmost position of protocol header
    AxR        = AxL + protWidth + 1;       // Rightost position of protocol header
if (continu)
    AyT=AyB;
else  AyT      = nPaintBeg-nVscrollPos;  // + rowHeight/2;    // Topmost position of protocol
header
    AyB        = AyT + protHeight + 1;      // Lowermost position of protocol header
if ( ((AyT+cyChar) > nPaintBeg) && (datalines<2))
    {
    TextOut(hdc, WORD(AxL), WORD((AyT+cyChar)),     //+yB-2*cyChar)/2),
            (LPCSTR)protocol_name, (WORD)(name_length-1)); // WORD(ptl->pnamelen()));
    AyT+=(3*cyChar);
```

```
    AyB+=(3*cyChar);
    }
  else
    {
    TextOut(hdc, WORD(AxL-3*cxChar), WORD((AyT+cyChar)), (LPCSTR)(itoa(datalines,buffer,10)),
      (WORD)strlen(buffer));
    }
  }

  void field::OutlineField(HDC hdc, protocol *p, U32 dspbW) const
  {
  HBRUSH hbr, hbrOld;
  RECT r;
  //
  // Compute top of rectangle defining this field
  //
  if ((yB < nPaintBeg) || (yT > nPaintEnd)) return;          // If out off scroll range, return
  r.top  = WORD(((fdwoff*8 + fshl)/dspbW)*rowHeight + yT);  // Topmost position of field
  if (r.top > nPaintEnd) return;
  r.bottom= WORD(r.top + rowHeight + 1);                    // Bottommost position of field
  if (r.bottom < nPaintBeg) return;
  r.left  = WORD(((fshl+fdwoff*8)%dspbW)*bitWidth + xL);    // Leftmost position of field
  r.right = WORD((32 - fshr)*bitWidth + r.left + 1);        // Rightmost position of field
  //
  // Draw Rectangle around protocol field; Display Field Name; Display Value
  //
  SetBkColor(hdc, (this == p->fieldptr(p->curfield())) ? RGB(192,192,192) : RGB(255,255,255));
  SetTextColor(hdc, RGB(0,0,0));
  hbr  =  CreateSolidBrush((this  ==  p->fieldptr(p->curfield()))  ?  RGB(192,192,192)  :
  RGB(255,255,255));
  hbrOld = SelectObject(hdc, hbr);
  if (r.right <= xR)
    Rectangle(hdc, r.left, r.top, r.right, r.bottom);
  else
    {
    S32 tmp = r.right - xR + 1;
    r.right = (WORD)xR;
```

```
    Rectangle(hdc, r.left, r.top, r.right, r.bottom);
    ++r.left; ++r.top; --r.bottom;
    FillRect(hdc, &r, hbr);
    r.top = r.bottom;
    if (r.top > nPaintEnd) return;
    r.bottom = WORD(r.top + rowHeight + 1);
    if (r.bottom < nPaintBeg) return;
    r.left = (WORD)xL;
    r.right = (WORD)(tmp + r.left);
    Rectangle(hdc, r.left, r.top, r.right, r.bottom);
    ++r.top; --r.bottom; --r.right;
    FillRect(hdc, &r, hbr);
    }
  SetTextAlign(hdc, TA_CENTER);
  ++r.top;   ++r.left;   --r.bottom; --r.right;
  if (fname != 0)
    ExtTextOut(hdc, WORD((r.left+r.right)/2),
          WORD((r.top+r.bottom-cyChar)/2),
          ETO_CLIPPED, (LPRECT)&r, (LPSTR)fname, (WORD)strlen(fname), (LPINT)0);
  /*
   * Restore original Brush and release resources
   */
  hbr = SelectObject(hdc, hbrOld);
  DeleteObject(hbr);
  }

  void field::OutlineFieldVal(HDC hdc, protocol *p, U32 dspbW, unsigned long val, unsigned char fmat,
  unsigned long bitwid) const
  {
  HBRUSH hbr, hbrOld;
  RECT r;
  char buffer[33];
  char tbuffer[33];
  //TEST
  //return;
  p=p;
  //
```

```
// Compute top of rectangle defining this field
//
if ((AyB < nPaintBeg) || (AyT > nPaintEnd)) return;          // If out off scroll range, return
r.top  = WORD(((fdwoff*8 + fshl)/dspbW)*rowHeight + AyT);  // Topmost position of field
if (r.top > nPaintEnd) return;
r.bottom= WORD(r.top + rowHeight + 1);                      // Bottommost position of field
if (r.bottom < nPaintBeg) return;

r.left  = WORD(((fshl+fdwoff*8)%dspbW)*AbitWidth + AxL);    // Leftmost position of field
r.right = WORD((32 - fshr)*AbitWidth + r.left + 1);         // Rightmost position of field
//
// Draw Rectangle around protocol field; Display Field Name; Display Value
//
//SetBkColor(hdc, (this == p->fieldptr(p->curfield())) ? RGB(192,192,192) : RGB(255,255,255));
//SetTextColor(hdc, RGB(0,0,0));
//hbr  =  CreateSolidBrush((this  ==  p->fieldptr(p->curfield()))  ?  RGB(192,192,192)  :
RGB(255,255,255));
hbrOld = SelectObject(hdc, hbr);
if (r.right <= AxR)
  Rectangle(hdc, r.left, r.top, r.right, r.bottom);
else
  {
  S32 tmp = r.right - AxR + 1;
  r.right = (WORD)AxR;
  Rectangle(hdc, r.left, r.top, r.right, r.bottom);
  ++r.left; ++r.top; --r.bottom;
  FillRect(hdc, &r, hbr);
  r.top = r.bottom;
  if (r.top > nPaintEnd) return;
  r.bottom = WORD(r.top + rowHeight + 1);
  if (r.bottom < nPaintBeg) return;
  r.left = (WORD)AxL;
  r.right = (WORD)(tmp + r.left);
  Rectangle(hdc, r.left, r.top, r.right, r.bottom);
  ++r.top; --r.bottom; --r.right;
  FillRect(hdc, &r, hbr);
  }
```

```
SetTextAlign(hdc, TA_CENTER);
++r.top;   ++r.left;  --r.bottom; --r.right;

if (fname != 0)
ExtTextOut(hdc, WORD((r.left+r.right)/2),
          WORD((r.top+r.bottom-2*cyChar)/2),
          ETO_CLIPPED, (LPRECT)&r, (LPSTR)fname, (WORD)strlen(fname), (LPINT)0);
ltoa(val,tbuffer,fmat==1 ? 10:16);
// All this to get zeros to print - something better??
if ((bitwid+3)/4 > strlen(tbuffer))
  {
  strcpy(buffer, "0 ");
  for (int i=1; i<((bitwid+3)/4-strlen(tbuffer)); i++)
    strcpy(strrchr(buffer, '0')+i,"0");
  strcpy(strrchr(buffer, '0')+1, tbuffer);
//  if ((val == 0) &&
//      strcpy(strrchr(buffer, '0')+1, "0");
  }
else strcpy(buffer, tbuffer);

ExtTextOut(hdc, WORD((r.left+r.right)/2),
          WORD((r.top+r.bottom)/2),
//          ETO_CLIPPED, (LPRECT)&r, (LPSTR)(), (WORD)(4), (LPINT)0);
          ETO_CLIPPED, (LPRECT)&r, (LPSTR)(buffer), (WORD)(strlen(buffer)), (LPINT)0);
/*
 * Restore original Brush and release resources
 */
hbr = SelectObject(hdc, hbrOld);
DeleteObject(hbr);
}

void protocol::FindField(HWND hwnd, U32 x, U32 y)
{

for (U32 i=0; i<num_fields; i++)
  {
  if (fs[i].FindF(x, y, dbW) && i != cur_field)
```

```
        {
        cur_field = (U16)i;
        InvalidateRect(hwnd, NULL, TRUE);
        return;
        }
      }
    }

    U32 field::FindF(U32 x, U32 y, U32 dspbW) const
    {

    RECT r;
      r.top  = WORD(((fdwoff*8 + fshl)/dspbW)*rowHeight + yT);  // Topmost position of field
      if (r.top > nPaintEnd) return(FALSE);
      r.bottom= WORD(r.top + rowHeight + 1);                    // Bottommost position of field
      if (r.bottom < nPaintBeg) return(FALSE);
      r.left  = WORD(((fshl+fdwoff*8)%dspbW)*bitWidth + xL);    // Leftmost position of field
      r.right = WORD((32 - fshr)*bitWidth + r.left + 1);        // Rightmost position of field
      return((x >= r.left) && (x <= r.right) && (y >=r.top) && (y <= r.bottom));
    }
    ////////////////////////////////
    void protocol::OutlineProtocol(HDC hdc) const
    {

    for (U32 i=0; i<num_fields; i++)
    //   if (i != cur_field && fs[i].bitlen())
      if (i != cur_field)     // && fs[i].bitlen())
        fs[i].OutlineField(hdc, (protocol *)this, dbW);
    fs[cur_field].OutlineField(hdc, (protocol *)this, dbW);
    }

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

```
////////////////////////////////////////////////////////////////////////
// Parse Fields
////////////////////////////////////////////////////////////////////////
static protocol *ParseFields2(protocol *p, U32 &HeaderLen)
{
protocol *LocalProto=0;
field *f, *g;

ProtoParseLen = 0;
HeaderLen = p->numbits();
f = g = p->fieldptr();
for (U32 i = 0;
     i < p->numfields() && ProtoParseLen < HeaderLen && ParseLen < FrameLen;
     f = &g[i])
  {
//
// Retrieve Current Field Value
//
  U32 val = f->get_value();
//
// If there is a filter criteria associated with the current field...
//    apply it to the frame
//
  if (f->fltptr() != 0)
    f->fltptr()->ApplyFilterCriteria(val);
//
// If the current field value is legal...
//
  if (f->value_ok(val, i, LocalProto))
    {
//
//    If the current field contains a checksum value...
//       verify it ... and if incorrect do something ?
//
    if (f->csumptr() != 0)
      f->csumptr()->verify();
//
```

```
    //    If the current field needs statistics collected...collect them
    //
          if (f->statsptr() != 0)
            f->statsptr()->collect(val, HeaderLen);
    //
    // If the current field contains routing information...
    //    lookup the specified routing information
    //    (Only use first routing protocol found to allow statistics gathering
    //     and filtering to be performed on every field in the frame)
    //
          if (RtePtr == 0 && f->rteptr())
            RtePtr = f->rteptr()->RouteFrame();
    //
    //    If the current field contains the protocol header length...
    //       extract and save it in global variable HeaderLen
    //
          if (f->hdlen())
            HeaderLen = val*f->hdlen();
    //
    //    Update # of bits parsed in the current frame (ParseLen)
    //    Update # of bits parsed in the current protocol header(ProtoParseLen)
    //
          ParseLen += f->bitlen();
          ProtoParseLen += f->bitlen();
    //
    //    If current field contains frame len (from start of current protocol)...
    //       extract and save it in global variable FrameLen
    //       If length extracted is larger than hardware frame length...
    //          this is a malformed network frame...do device specific response
    //
          if (f->protlen()) // will be val*f->protlen() + ... when done with intf
            FrameLen = min(HwLen, (val*8 + ParseLen - ProtoParseLen));
    #ifndef notdef
          f->OutlineFieldVal(hdc, p, p->dspbW(), val, f->format(), f->bitlen());
    #endif
        }
      }
```

```
    //
    // Check required to skip over unsupported/unparsed options
    //
    if (LocalProto == 0 && HeaderLen > ProtoParseLen) // Not all Options Parsed
      {                                  // so skip over them
      ParseLen += (HeaderLen - ProtoParseLen);
      ProtoParseLen = HeaderLen;
      }
    ParsePtr += ProtoParseLen/8;
    return(LocalProto);
    }

    ////////////////////////////////////////////////////////////////////////
    // Parse Protocol
    ////////////////////////////////////////////////////////////////////////
    protocol *ParseProtocol2(protocol *p)
    {
    //
    // Parse each field in the protocol header
    //
    protocol *LocalProto = ParseFields2(p, HeaderLen);
    if (ParseLen >= HwLen) return(0); // Finished if all hardware bits are parsed
    //
    // If (this protocol supports optional fields and...
    //     this protocol header contains one or more optional fields)
    //     Process each option bit contained in the protocol header
    //
    if ((p = p->options()) != 0)
      {  // this protocol supports optional fields
      while(ProtoParseLen < HeaderLen)
        {// There are (HeaderLen-ProtoParseLen) bits of optional data remaining
        U32 optlen;
        HeaderLen -= ProtoParseLen;
        p = ParseFields2(p, optlen);
        }
      if (ParseLen >= HwLen) return(0);// Finished if all hardware bits are parsed
      }
```

```
    //
    // return next protocol to use in parsing this frame
    //
    return(LocalProto);
}

////////////////////////////////////////////////////////////////////////
// ParseFrame(protocol *InitialProtocolPtr);
//    Parameters: p - pointer to initial protocol description object
//
//    Globals must be set as follows:
//        HwLen   - set to number of bits received from interface device
//        FramePtr- set to start of frame received from interface device
//        SrcIntf - set to indicate the interface number which received frame
////////////////////////////////////////////////////////////////////////
void ParseFrame2(protocol *ProtPtr)
{
RtePtr    = 0;      // Clear Routing Table Entry Pointer at start of frame
ParseLen  = 0;      // Set # of Bits Parsed in Frame to 0 at start of frame
ParseLevel = 0;     // ISO 7 layer position indication
FrameLen  = HwLen;  // Set Frame Length to Hardware Length as best guess
ParsePtr  = FramePtr; // Set ParsePtr to point at start of Frame
//
////////Update Number of Frames and Bits Received
//
// Parse each protocol in the Frame (Save pointer to protocol header first)
//
while(ProtPtr != 0)
  {
  ParseList.operator[](ParseLevel++) = ParsePtr;  // Save header pointer
//
  ProtPtr = ParseProtocol2(ProtPtr);
  }
//
// If there is data...parse it
//
while(ParseLen < FrameLen) ParseProtocol2(DataPtlPtr);
```

```
    //
    // If there is pad...parse it
    //
    FrameLen = HwLen;
    while(ParseLen < FrameLen) ParseProtocol2(PadPtlPtr);
    //
    // If Frame passed at least one configured filter...Determine where to send it
    //
    if (CfgFilters.FrameFilterStatus() == PASS_FRAME)
      {
      if (RtePtr == 0 || RtePtr->RouteFrame() == 0)
        // Increment number of slow-path frames
        ; // Send frame up to software
      else
        // Increment number of fast-path frames
        ; // Send frame to designated interface
      // Increment number of frames passed
      }
    else
      {
      ///////Update Number of Frames and Bits Filtered
      }
    CfgFilters.reset();    // Reset criteria in each onfigured filter
    }
/*************************************************************************
****/
/*************************************************************************
****/
/*************************************************************************
****/
/*************************************************************************
****/
static protocol *ParseFields(protocol *p, U32 &HeaderLen)
{
protocol *tp=0;
field *f, *g;
```

```
    ProtoParseLen = 0;
    HeaderLen = p->numbits();
    f = g = p->fieldptr();
    for (U32 i=0; i<p->numfields() && ProtoParseLen < HeaderLen; f = &g[i])
5     {
      unsigned long val = f->get_value();       //
      if (f->swap())
        val = wordswap(val);
      if (f->value_ok(val, i, tp))
10      {
        if (f->csumptr() != 0)
          f->csumptr()->verify();
        if (f->statsptr() != 0)            // If the user has requested stats be kept on this field
          f->statsptr()->collect(val, HeaderLen); //    collect it in whatever format has been requested
15  //
        if (f->hdlen())                    // If this field contains the protocol header length
          HeaderLen = val*f->hdlen();          //    save it for determining option length
    //
        ParseLen += f->bitlen();           // Keep total number of packet bits parsed
20      ProtoParseLen += f->bitlen();          // Keep total number of protocol bits parsed
        if (f->protlen())
          FrameLen = val*8 + ParseLen - ProtoParseLen;
    // Need to return an error value, otherwise, e.g., if IP with bad len returns && we crash
    //trying to parse options....
25      f->OutlineFieldVal(hdc, p, p->dspbW(), val, f->format(), f->bitlen());
        if (ParseLen >= HwLen) return(0);
        }
      }
    ParsePtr += ProtoParseLen/8;
30  return(tp);
    }
    /*************************************************************************
    ****/
    protocol *ParseFrame(protocol *p)
35  {
    unsigned long HeaderLen;
    protocol *tp=0;
```

```
tp = ParseFields(p, HeaderLen);              // Parse each field in the basic protocol
//
// Process options until all option bytes
//
if ( (FrameLen <= HwLen) && (ParseLen <= HwLen) )   // these are the error returns from parse
fields
  {                                  // returns - do something better with these
  if ((p = p->options()) != 0)           // If this protocol has any optional fields and ...
    while(ProtoParseLen < HeaderLen)           // there are some in this packet
      {                              // process them
      unsigned long optlen;
      HeaderLen -= ProtoParseLen;
      p = ParseFields(p, optlen);
      }
  }
else                                 // do something with the error cases
  {
  if (ParseLen > HwLen) return(0);
  FrameLen = HwLen;
  }
if (tp == 0)
  {
  if (FrameLen <= HwLen)
    {
    protocols data("DATA", 0), t;
    ProtocolList->find(data, t);
    while (ParseLen < FrameLen)
      {
      datalines++;
      t.prot()->setupOutlineCoordsMult(TRUE);
      ParseFields(t.prot(), HeaderLen);
      }
    }
  protocols pad("PAD", 0), t;
  ProtocolList->find(pad, t);
  while (ParseLen < HwLen)           // Pad
    {
```

```
        padlines++;
        t.prot()->setupOutlineCoordsMult(TRUE);
        ParseFields(t.prot(), HeaderLen);
        }
    return(0);
    }
  else
    return(tp);
  }
→

////////////////////////////////////////////////////////////////////////
//    FILTER.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"
////////////////////////////////////////////////////////////////////////
// Channel Class Functions
//    Destructor: ~channel
////////////////////////////////////////////////////////////////////////
//
channel::~channel()
  {
  if (Criteria != 0)    delete []Criteria;

  if (ChannelName != 0) delete []ChannelName;
  }
//
void channel::Update(verify *v)
  {
  NextCriteriaIndex = v->nxtidx;
  if (v->prot == (protocol *)PASS_FRAME)
    {
    ++FramesAccepted;
    FrameBitsAccepted += HwLen;
    CfgFilters.FrameFilterStatus(PASS_FRAME); // Set Status to PASS
    }
```

```
    }

////////////////////////////////////////////////////////////////////
// criteria Class Functions
//    ApplyFilterCriteria
////////////////////////////////////////////////////////////////////
//
// Applies the configured filter criteria to a received field value
//    If index of this criteria == index of the next expected criteria
//       lookup the value and update the associated filter
//    If index of this criteria > index of the next expected criteria
//       disable further application of this filter
//
void criteria::ApplyFilterCriteria(U32 value)
   {
   if (Index == ChPtr->NciValue())          // Next criteria to evaluate ?
      ChPtr->Update(Ranges->value_ok(value));
   else
      if (Index > ChPtr->NciValue())        // Filter Criteria skipped ?
         ChPtr->Disable();                  // Disable filter evaluation
//   if (Index < ChPtr->NciValue())          // Skip this criteria ?
   }


////////////////////////////////////////////////////////////////////
//    LOOKUP.CPP
////////////////////////////////////////////////////////////////////

#include "gen.hpp"

void verify::out_to_file(FILE *fp) const
{
U32 tmp;

if (prot == 0 || prot->pname() == 0)
   {
   tmp = 0;
```

```
    fwrite(&tmp,        sizeof(tmp),   1,  fp);
    }
  else
    {
    tmp = prot->pnamelen();
    fwrite(&tmp,        sizeof(tmp),   1,  fp);
    fwrite(prot->pname(),  tmp,        1,  fp);
    prot->out_to_file();
    }
fwrite(&nxtidx,      sizeof(nxtidx), 1,  fp);
fwrite(&minval,      sizeof(minval), 1,  fp);
fwrite(&maxval,      sizeof(maxval), 1,  fp);
fwrite(&okbits,      sizeof(okbits), 1,  fp);
}
//
//
//
void verify::get_from_file(FILE *fp)
{
protocols psearch(fp), t;              // Because destructor does not delete pname memory will be lost
if ( (ProtocolList != NULL) && (ProtocolList->find(psearch, t) != 0))
  prot = t.prot();
else
  prot = 0;
//
// Read remainder of verify structure (relates value/range to protocol)
//
fread(&nxtidx,       sizeof(nxtidx), 1,  fp);
fread(&minval,       sizeof(minval), 1,  fp);
fread(&maxval,       sizeof(maxval), 1,  fp);
fread(&okbits,       sizeof(okbits), 1,  fp);
}
//
//
//
lookup *alloc_lookup_structs(U32 type, field *f)
{
```

```
    switch(type)
      {
      default:      cout << "LOOKUP Yipes!!!\n";
      case NOLOOKUP: return(0);                    // No p
      case ONEVALUE: return(new lookup_valu());
      case ARRAY:    return(new lookup_vect(1 << (32-f-> shrbits())));
      case TREE:     return(new lookup_tree());
      }
    }

lookup *alloc_lookup_tree()
{
return(new lookup_tree());
// allocs WCDEFAULT_VECTOR_LENGTH which is 10
}

////////////////////////////////////////////////////////////////////////
//    MAIN.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"
#define NUMLINES 4800
#define min(a,b) (((a) < (b)) ? (a) : (b))
#define max(a,b) (((a) > (b)) ? (a) : (b))
S32       nVscrollInc, nHscrollInc ;
// TEMP
extern int datalines;
extern int padlines;

// TEMP

#define UNTITLED "(untitled)"
      // Functions in POPFILE.C make such for pa, use this for now
static S32    nMaxWidth,  nVscrollMax, nHscrollPos, nHscrollMax ;

void PopFileInitializeTr (HWND);
void PopFileInitialize (HWND, char *) ;
```

```
BOOL PopFileOpenDlg   (HWND, LPSTR, LPSTR);
BOOL PopFileSaveDlg   (HWND, LPSTR, LPSTR);
protocol *PopFileRead  (HWND, LPSTR);
//BOOL PopFileWrite    (HWND, LPSTR);
BOOL            NAMEOK100;
BOOL            BITLENOK100;
BOOL            OFFSETOK100;
BOOL            ANLZ_OK100;
    S32 cxClient, cyClient, cxChar, cyChar, nVscrollPos, nPaintBeg, nPaintEnd;
long FAR PASCAL _export FrameWndProc  (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL _export CloseEnumProc (HWND, LONG);
long FAR PASCAL _export ImageWndProc  (HWND, UINT, UINT, LONG);
long FAR PASCAL _export ConfigWndProc  (HWND, UINT, UINT, LONG);
long FAR PASCAL _export NxtPtlWndProc  (HWND, UINT, UINT, LONG);
long FAR PASCAL _export DisplayImageWndProc  (HWND, UINT, UINT, LONG);
long FAR PASCAL _export AnlzWndProc  (HWND, UINT, UINT, LONG);

    // global variables

char  szFrameClass [] = "MdiFrame";
char  szImageClass [] = "Protocol Image";
char  szConfigClass[] = "PtlCfg";
char  szNxtPtlClass[] = "NxtPtlCfg";
char  szFilterClass[] = "FILTER";
char  szDImageClass [] = "Frame Image";
char  szAnlzClass[] = "ANLZ";
HANDLE hInst;
HMENU  hMenu;
HMENU  hMenuWindow;
static HWND hDlgModeless;
static char szAppName[] = "NbPa";
// Temp Temp
S8 *savepkt;
S8 *savemalloc;
HDC         hdc;
// Temp Temp
```

```
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
        LPSTR lpszCmdLine, WORD nCmdShow)
{
HANDLE  hAccel;
HWND    hwndFrame, hwndClient;
MSG     msg;
WNDCLASS wndclass;

hInst = hInstance;
lpszCmdLine = 0; // Eliminate compiler warning

if (!hPrevInstance)
  {
        // Register the frame window class

  wndclass.style       = CS_HREDRAW | CS_VREDRAW;
  wndclass.lpfnWndProc  = (WNDPROC)FrameWndProc;
  wndclass.cbClsExtra   = 0;
  wndclass.cbWndExtra   = 0;
  wndclass.hInstance    = hInstance;
  wndclass.hIcon        = LoadIcon (hInstance, "ICON_2"); // IDI_APPLICATION);
  wndclass.hCursor      = LoadCursor (NULL, IDC_ARROW);
  wndclass.hbrBackground = COLOR_APPWORKSPACE + 1;
  wndclass.lpszMenuName  = NULL;
  wndclass.lpszClassName = szFrameClass;

  RegisterClass (&wndclass);


        // Register the Image child window class

  wndclass.style       = CS_HREDRAW | CS_VREDRAW;
  wndclass.lpfnWndProc  = (WNDPROC)ImageWndProc;
  wndclass.cbClsExtra   = 0;
  wndclass.cbWndExtra   = sizeof (LONG);
  wndclass.hInstance    = hInstance;
  wndclass.hIcon        = LoadIcon (hInstance, "ICON_1"); //IDI_APPLICATION);
```

```
wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = COLOR_APPWORKSPACE + 1;
wndclass.lpszMenuName  = NULL ;
wndclass.lpszClassName = szImageClass ;

RegisterClass (&wndclass) ;


        // Register the Configuration child window class

wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc   = (WNDPROC)ConfigWndProc ;
wndclass.cbClsExtra    = 0 ;
wndclass.cbWndExtra    = 0 ;
wndclass.hInstance     = hInstance ;
wndclass.hIcon         = NULL ;
wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = GetStockObject (LTGRAY_BRUSH) ;
wndclass.lpszMenuName  = NULL ;
wndclass.lpszClassName = szConfigClass ;

RegisterClass (&wndclass) ;


        // Register the Next Protocol Configuration grandchild window class

wndclass.style         = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc   = (WNDPROC)NxtPtlWndProc ;
wndclass.cbClsExtra    = 0 ;
wndclass.cbWndExtra    = 0 ;
wndclass.hInstance     = hInstance ;
wndclass.hIcon         = NULL ;
wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = GetStockObject (LTGRAY_BRUSH) ;
wndclass.lpszMenuName  = NULL ;
wndclass.lpszClassName = szNxtPtlClass ;
```

```
RegisterClass (&wndclass);


        // Register the Display Image child window class

wndclass.style      = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc  = (WNDPROC)DisplayImageWndProc;
wndclass.cbClsExtra   = 0;
wndclass.cbWndExtra   = sizeof (LONG);
wndclass.hInstance    = hInstance;
wndclass.hIcon      = LoadIcon (hInstance, "ICON_1"); //IDI_APPLICATION);
wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground = COLOR_APPWORKSPACE + 1;
wndclass.lpszMenuName  = NULL;
wndclass.lpszClassName = szDImageClass;

RegisterClass (&wndclass);

        // Register the ANLZ dialog box child window class

wndclass.style      = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc  = (WNDPROC)AnlzWndProc;
wndclass.cbClsExtra   = 0;
wndclass.cbWndExtra   = 0;
wndclass.hInstance    = hInstance;
wndclass.hIcon      = NULL;
wndclass.hCursor     = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground = GetStockObject (LTGRAY_BRUSH);
wndclass.lpszMenuName  = NULL;
wndclass.lpszClassName = szAnlzClass;

RegisterClass (&wndclass);
}
        // Get all filter channel info
// setup_filters();
        // Get all protocol info
```

```
setup_protocols();

        // Obtain handles to three possible menus & submenus

hMenu     = LoadMenu (hInst, "MdiMenuMain");
hMenuWindow= GetSubMenu (hMenu,  MAIN_MENU_POS);

        // Load accelerator table

hAccel = LoadAccelerators (hInst, "MdiAccel");

        // Create the frame window

hwndFrame = CreateWindow (szFrameClass, "NbPa",
                WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                0, 0,
                GetSystemMetrics(SM_CXSCREEN), GetSystemMetrics(SM_CYSCREEN),
                NULL, hMenu, hInstance, NULL);

hwndClient = GetWindow (hwndFrame, GW_CHILD);

ShowWindow (hwndFrame, nCmdShow);
UpdateWindow (hwndFrame);

        // Enter the modified message loop

//while (GetMessage (&msg, NULL, WM_KEYFIRST, WM_KEYLAST))
while (GetMessage (&msg, NULL, 0, 0))
  {
  if (hDlgModeless == 0 || !IsDialogMessage(hDlgModeless, &msg))
    {
    if (!TranslateMDISysAccel (hwndClient, &msg) &&
      !TranslateAccelerator (hwndFrame, hAccel, &msg))
      {
      TranslateMessage (&msg);
      DispatchMessage (&msg);
      }
```

```
        }
      }
    return msg.wParam ;
    }
////////////////////////////////////
#ifndef notdef
void DoCaption (HWND hwnd, char *szTitleName)
    {
    SetWindowText (hwnd, (szTitleName [0] ? szTitleName : UNTITLED)) ;
    }

void OkMessage (HWND hwnd, char *szMessage, char *szTitleName)
    {
    char szBuffer [64 + _MAX_FNAME + _MAX_EXT] ;

    wsprintf (szBuffer, szMessage,
            (LPSTR) (szTitleName [0] ? szTitleName : UNTITLED)) ;

    MessageBox (hwnd, szBuffer, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    }

short AskAboutSave (HWND hwnd, char *szTitleName)
    {
    char  szBuffer [64 + _MAX_FNAME + _MAX_EXT] ;
    short nReturn ;

    wsprintf (szBuffer, "Save current changes in %s?",
            (LPSTR) (szTitleName [0] ? szTitleName : UNTITLED)) ;

    nReturn = MessageBox (hwnd, szBuffer, szAppName,
                MB_YESNOCANCEL | MB_ICONQUESTION) ;

    if (nReturn == IDYES)
        if (!SendMessage (hwnd, WM_COMMAND, IDM_SAVE, 0L))
            nReturn = IDCANCEL ;

    return nReturn ;
```

```
    }
#endif
// extern S32 xL, xR, yT, yB;
static U32 soughtindex = 0;
static U16 total=0;

////////////////////////////////
long FAR PASCAL _export FrameWndProc (HWND hwnd, UINT message, UINT wParam, LONG lParam)
{
RECT r;
static HWND      hwndClient ;
CLIENTCREATESTRUCT clientcreate ;
FARPROC          lpfnEnum ;
HWND             hwndChild1 ;
MDICREATESTRUCT   mdicreate ;
static protocol *tptl;
static protocol *aptl=0;          // pointer to base MAC ptl: length field not flagged
static protocol *fptl=0;          // pointer to trace file format ptl: length field used
                          // to set FrameLen && HwLen and skip balance of frame
static U32 sought = 0;

int i;
static BOOL   bNeedSave = FALSE ;
static char   szFileName [_MAX_PATH] ;
static char   szTitleName [_MAX_FNAME + _MAX_EXT] ;
static char   TraceFileName [_MAX_PATH] ;  //??
static char   TraceTitleName [_MAX_FNAME + _MAX_EXT] ; // ??
static char   tmpszFileName [_MAX_PATH] ; // find something better
FILE *fp;
protocol  *ptl=0;
protocol *lptl=NULL;

    switch (message)
        {
        case WM_CREATE:       // Create the client window
          TraceFileName [0] = TraceTitleName [0] = '\0' ; // some other place?
```

```
            clientcreate.hWindowMenu  = hMenuWindow ;
            clientcreate.idFirstChild = IDM_FIRSTCHILD ;

            hwndClient = CreateWindow ("MDICLIENT", NULL,
                              WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE | WS_VSCROLL |
 5  WS_HSCROLL,
                        0, 0, 0, 0, hwnd, 1, hInst,
                        (LPSTR) &clientcreate) ;
            return 0 ;

          case WM_COMMAND:
10          switch (wParam)
              {
                    // Messages from File menu

              case IDM_NEW:
                PopFileInitialize (hwnd, "*.PDF") ;
15              if (bNeedSave && IDCANCEL ==
                      AskAboutSave (hwnd, szTitleName))
                    return 0 ;
                bNeedSave = FALSE ;  // ??
                if (PopFileOpenDlg (hwnd, szFileName, szTitleName))
20                {
                  if ((tptl = PopFileRead (hwnd, szFileName)) == 0)
                    {
                    if (0 == strrchr(szFileName, '.'))
                      strcpy(strrchr(szFileName, ' '), ".PDF");
25                  else strcpy(strrchr(szFileName, '.'), ".PDF"); /* change extension */

                    if ((tptl = setup_newprotocol(szFileName))==FALSE)
                        return(0);
                    }
                  GetClientRect(hwndClient, (LPRECT)&r);

30                mdicreate.szClass = szImageClass;
                  mdicreate.szTitle = tptl->pname();
                  mdicreate.hOwner  = hInst;
```

```
                mdicreate.x      = r.left;
                mdicreate.y      = r.top;
                mdicreate.cx     = WORD(r.right - r.left);
                mdicreate.cy     = WORD((r.bottom - r.top) / 1);
                mdicreate.style  = 0;
                mdicreate.lParam = (LONG)tptl;
                hwndChild1       = (HWND)SendMessage (hwndClient, WM_MDICREATE, 0,
                                        (long) (LPMDICREATESTRUCT) &mdicreate);
                NAMEOK100=FALSE;      // indicate first sight of PTL_NAME
                BITLENOK100=FALSE;    // indicate first sight of FIELD_BITLEN
                OFFSETOK100=FALSE;    // indicate first sight of FIELD_OFFSET
                hDlgModeless = CreateDialogParam(hInst, szConfigClass, hwndChild1,
                        MakeProcInstance ((FARPROC) ConfigWndProc, hInst), 0);
                }
              return 0 ;

            case IDM_OPEN:
              PopFileInitialize (hwnd, "*.PDF") ;

              if (bNeedSave && IDCANCEL == AskAboutSave (hwnd, szTitleName))
                return 0 ;

              if (PopFileOpenDlg (hwnd, szFileName, szTitleName))
                {
                if ((tptl = PopFileRead (hwnd, szFileName)) == 0)
                  {
                  OkMessage (hwnd, "Could not read protocol template file %s!", szTitleName) ;
                  szFileName [0] = szTitleName [0] = '\0' ;
                  return(0);
                  }
                if (tptl->fieldptr()==NULL)
                  {
                  OkMessage (hwnd,
                "Null Field Pointer - Could not read protocol template file %s",
                    tptl->filename() ) ;
                  return (0);
                  }
```

```
                GetClientRect(hwndClient, (LPRECT)&r);

                mdicreate.szClass = szImageClass;
                mdicreate.szTitle = tptl->pname();
                mdicreate.hOwner  = hInst;
                mdicreate.x       = r.left;
                mdicreate.y       = r.top;
                mdicreate.cx      = WORD(r.right - r.left);
                mdicreate.cy      = WORD((r.bottom - r.top) / 1);
                mdicreate.style   = 0;
                mdicreate.lParam  = (LONG)tptl;
                hwndChild1        = (HWND)SendMessage (hwndClient, WM_MDICREATE, 0,
                                        (long) (LPMDICREATESTRUCT) &mdicreate);
                NAMEOK100=FALSE;      // indicate first sight of PTL_NAME
                BITLENOK100=FALSE;    // indicate first sight of FIELD_BITLEN
                OFFSETOK100=FALSE;    // indicate first sight of FIELD_OFFSET
                hDlgModeless = CreateDialogParam(hInst, szConfigClass, hwndChild1,
                        MakeProcInstance ((FARPROC) ConfigWndProc, hInst), 0);
                }
              bNeedSave = FALSE ;
              return 0 ;

            case IDM_SAVE:
                if (szFileName[0])
                  {
                  if (0==strrchr( szFileName, '\\'))
                    strcpy(tmpszFileName, szFileName);
                  else strcpy(tmpszFileName, strrchr(szFileName, '\\')+1  );
                  for ( i=0; i<ProtocolList->entries(); i++)
                    {
                    if  (strcmp( tmpszFileName,
                          ProtocolList->operator[](i).prot()->filename()) == 0)
                        {
ProtocolList->operator[](i).prot()->clear_out_flag();
                        ProtocolList->operator[](i).prot()->out_to_file();
ProtocolList->operator[](i).prot()->clear_out_flag();
                        bNeedSave = FALSE ;
```

```
                    return 1 ;
                    }
                  }
                OkMessage(hwnd, "Could not write file %s", szTitleName) ;
                return 0 ;
                }
                  // Fall through

            case IDM_SAVEAS:
              if (PopFileSaveDlg (hwnd, szFileName, szTitleName))
                {
                DoCaption (hwnd, szTitleName) ;
                for ( i=0; i < ProtocolList->entries(); i++)
                  {
                  if  (strcmp(tptl->pname(),
                     ProtocolList->operator[](i).prot()->pname()) == 0)
                  break;
                  }
    //
    // Finds last file opened && writes current values for that entry
    //
                if (i==ProtocolList->entries())
                  {
                  OkMessage (hwnd, "Could not write file %s", szTitleName);
                  return(0);
                  }
                if (szFileName[0])
                  {
                  if (0==strrchr( szFileName, '\\'))
                    strcpy(tmpszFileName, szFileName);
                  else strcpy(tmpszFileName, strrchr(szFileName, '\\')+1 );
                  if ((fp = fopen(tmpszFileName, "wb"))==0) return (0);
    ProtocolList->operator[](i).prot()->clear_out_flag();
                  ProtocolList->operator[](i).prot()->out_to_file(fp);
    ProtocolList->operator[](i).prot()->clear_out_flag();
    //              bNeedSave = FALSE ;
                  return (1) ;
```

```
                    }
                  }
                return (0);

              case IDM_CLOSE:        // Close the active window

                hwndChild1 = LOWORD (SendMessage (hwndClient,
                                WM_MDIGETACTIVE, 0, 0L));

    //            if (SendMessage (hwndChild1, WM_QUERYENDSESSION, 0, 0L))
                  SendMessage (hwndClient, WM_MDIDESTROY,
                          hwndChild1, 0L);
                if (szFileName[0])       // Save the changes- make this a query??
                  {
                  if (0 == strrchr( szFileName, '\\'))
                    strcpy(tmpszFileName, szFileName);
                  else strcpy(tmpszFileName, strrchr(szFileName, '\\')+1 );
                  for ( i=0; i < ProtocolList->entries(); i++)
                    {
                    if (strcmp( tmpszFileName,
                          ProtocolList->operator[](i).prot()->filename()) == 0)
                      {
ProtocolList->operator[](i).prot()->clear_out_flag();
                      ProtocolList->operator[](i).prot()->out_to_file();
ProtocolList->operator[](i).prot()->clear_out_flag();
//
// Save all filter channels when CLOSE -
//
////////////////////////////////////////////////

////////////////////////////////////////
                      bNeedSave = FALSE;
                      return 1;
                      }
                    }
                  OkMessage(hwnd, "Could not write file %s", szTitleName);
                  }
```

```
                return 0 ;

            case IDM_EXIT:          // Exit the program
    for (i=0; i<ProtocolList->entries(); i++)
      ProtocolList->operator[](i).prot()->clear_out_flag();
    for ( i=0; i<ProtocolList->entries(); i++)
      ProtocolList->operator[](i).prot()->out_to_file();
    //
    // ? Check for duplicate protocol names, Ok/Cancel Message?
    //
                SendMessage (hwnd, WM_CLOSE, 0, 0L) ;
                return 0 ;

                    // Messages for arranging windows
            case IDM_TILE:
              SendMessage (hwndClient, WM_MDITILE, 0, 0L) ;

                return 0 ;

            case IDM_CASCADE:
                SendMessage (hwndClient, WM_MDICASCADE, 0, 0L) ;
                return 0 ;

            case IDM_ARRANGE:
                SendMessage (hwndClient, WM_MDIICONARRANGE, 0, 0L) ;
                return 0 ;

            case IDM_CLOSEALL:      // Attempt to close all children

                lpfnEnum = MakeProcInstance ((FARPROC) CloseEnumProc,
                                hInst) ;
                EnumChildWindows (hwndClient, lpfnEnum, 0L) ;
                FreeProcInstance (lpfnEnum) ;
                return 0 ;

            case IDM_MACPTL:
                PopFileInitialize (hwnd, "*.PDF") ;
```

```
if (PopFileOpenDlg (hwnd, szFileName, szTitleName))
  {
  if ((aptl = PopFileRead (hwnd, szFileName)) == 0)
    {
    OkMessage (hwnd, "Could not read protocol template file %s!", szTitleName) ;
    szFileName  [0] = szTitleName [0] = '\0' ;
    return(0);
    }
  if (aptl-> fieldptr()==NULL)
    {
    OkMessage (hwnd,
  "Null Field Pointer - Could not read protocol template file %s",
      aptl-> filename() ) ;
    return (0);
    }
  }
return (0);

case IDM_FRAMENAME:
  PopFileInitialize (hwnd, "*.PDF") ;
  if (PopFileOpenDlg (hwnd, szFileName, szTitleName))
    {
    if ((fptl = PopFileRead (hwnd, szFileName)) == 0)
      {
      OkMessage (hwnd, "Could not read protocol template file %s!", szTitleName) ;
      szFileName  [0] = szTitleName [0] = '\0' ;
      return(0);
      }
    if (fptl-> fieldptr()==NULL)
      {
      OkMessage (hwnd,
    "Null Field Pointer - Could not read protocol template file %s",
        fptl-> filename() ) ;
      return (0);
      }
    }
  sought = soughtindex = 0;
```

```
                return (0);

            case IDM_FRAMEFILE:
            case IDM_RUN:
            if (TraceFileName [0] != '\0')
                {
                hwndChild1 = LOWORD (SendMessage (hwndClient,
                        WM_MDIGETACTIVE, 0, 0L));

                SendMessage (hwndClient, WM_MDIDESTROY,
                        hwndChild1, 0L);
                TraceFileName [0] = TraceTitleName [0] = '\0';
                }

    //          if ((fptl == 0) || (aptl == NULL))
                if (aptl == NULL)
                 {
                 SendMessage (hwnd, WM_COMMAND, IDM_MACPTL, 0L);
                 if (aptl == NULL)
                   {
                   OkMessage (hwnd, "Could not read protocol template file %s!", szTitleName);
                   szFileName [0] = szTitleName [0] = '\0';
                   return(0);
                   }
                 }

                PopFileInitialize(hwnd, "*.TR1");
                if (PopFileOpenDlg (hwnd, TraceFileName, TraceTitleName))
                  {
    //
    // In Lanz .tr1 ether file: first frame record is 0xbca bytes into file. The frame record
    // format is 0x1005, followed by a length. This length + 4 bytes is the length of the frame
    // record. 36 bytes of this is Lanz header. tmps-32 = HwLen. tmps+4 == next fseek (from the
    // location at which current tmps is read).
    // Some files: begin at e0 rather than bca.
    // Delimiter is 10 xx followed by length to next delimiter from first byte after current
    // length. Network data delimiter is 10 05
```

```
// Try a search for first 10 05 in file??
//
                sought = 2;
                U16 tmps;
                int qq;
                if (fptl != 0)
                  {
                  while (strcmp( lptl->pname(), aptl->pname()) != 0)
                    {
                    FILE *fp = fopen(TraceFileName, "rb");
                    if (fp != 0)
                      {
                      qq = fseek(fp, sought-2, SEEK_SET);   // length field
                      if (qq==0)
                        {
                        fread(&tmps,   sizeof(tmps),   1,  fp);
                        if (tmps==0x1003)
                          {
                          fseek(fp, 4, SEEK_CUR);
                          fread(&total,   sizeof(total),   1,  fp);
                          fseek(fp, -6, SEEK_CUR);
                          }
                        qq = fseek(fp, sought, SEEK_SET);   // length field
                        if (qq==0)
                          {
                          fread(&tmps,   sizeof(tmps),   1,  fp);
                          if (tmps < 0xfff)            // some #
                            {
                            sought +=(tmps+4);     // position of next length field
                            qq = fseek(fp, -4, SEEK_CUR);   // start of frame
                            if (ParsePtr == NULL)
                              savemalloc = ParsePtr = (char *)malloc(0xfff);
                            else ParsePtr=savemalloc;
                            if (fread(ParsePtr, tmps,  1,  fp) != 0)
                              {
                              HwLen = FrameLen = (tmps+4) * 8;        // lanz issue, or work
w/ .pfds
```

```
                    fclose(fp);
                    lptl = ParseProtocol2(fptl);
                    if (lptl == NULL) break;
                    if (lptl == aptl)
                      {
                      ParsePtr-=4;
                      break;
                      }
                    }
                  }
                else
                  {
                        OkMessage (hwnd, "Could not read more frames %s!",
    TraceTitleName);
                    szFileName [0] = TraceTitleName [0] = '\0';
                    return(0);
                    }
                  }
                }
              fclose(fp);
              ParseLen = padlines = datalines = 0;
              }
            }
          }
        else
          {
          FILE *fp = fopen(TraceFileName, "rb");
          if (fp != 0)
            {
            qq = fseek(fp, sought, SEEK_SET);   // length field
            if (qq==0)
              {
              fread(&tmps,   sizeof(tmps),   1,  fp);
              if (tmps < 0xfff)            // some #
                {
                sought +=(tmps+4);     // position of next length field
                qq = fseek(fp, -4, SEEK_CUR);   // start of frame
```

```
                if (ParsePtr == NULL)
                  savemalloc = ParsePtr = (char *)malloc(0xfff);
                else ParsePtr=savemalloc;
                if (fread(ParsePtr, tmps,  1,  fp) != 0)
                  {
                 HwLen = FrameLen = (tmps+4) * 8;        // lanz issue, or work w/ .pfds
                  fclose(fp);
                  }
                }
              else
                {
                OkMessage (hwnd, "Could not read more frames %s!", TraceTitleName) ;
                szFileName  [0] = TraceTitleName [0] = '\0' ;
                return(0);
                }
              }
            }
          }
        soughtindex = 1;
        if (aptl == NULL)
          {
          OkMessage (hwnd, "No base protocol file selected %s", "  ") ;
          szFileName  [0] = szTitleName [0] = '\0' ;
          return(0);
          }
        GetClientRect(hwndClient, (LPRECT)&r);

        mdicreate.szClass = szDImageClass;
        mdicreate.szTitle = TraceTitleName;  //aptl->pname();
        mdicreate.hOwner  = hInst;
        mdicreate.x       = r.left;
        mdicreate.y       = r.top;
        mdicreate.cx      = WORD(r.right - r.left);
        mdicreate.cy      = WORD((r.bottom - r.top) / 1);
        mdicreate.style   = 0;
        mdicreate.lParam  = (LONG)aptl;
        hwndChild1        = (HWND)SendMessage (hwndClient, WM_MDICREATE, 0,
```

```
                                    (long) (LPMDICREATESTRUCT) &mdicreate);
            ANLZ_OK100=FALSE;   // indicate first sight of ANLZ_CUR change

            hDlgModeless     = CreateDialogParam(hInst, szAnlzClass, hwndChild1,
                        MakeProcInstance ((FARPROC) AnlzWndProc, hInst), 0);

            }
          return (0);

        case IDM_NEXTFRAME:

          if (aptl == 0)
            {
            OkMessage (hwnd, "Could not read protocol template file %s!", TraceTitleName) ;
            szFileName  [0] = TraceTitleName [0] = '\0' ;
            return(0);
            }
          else if (soughtindex > 0)
            {
            FILE *fp = fopen(TraceFileName, "rb");
            if ( fp != NULL)
              {
              int qq = fseek(fp, sought, SEEK_SET);   // length field
              U16 tmps;
              if (qq==0)
                {
                fread(&tmps,   sizeof(tmps),   1,  fp);
                if (tmps < 0xfff)            // some #
                  {
                  sought +=(tmps+4);     // position of next length field
                  qq = fseek(fp, -4, SEEK_CUR);   // start of frame
                  if (ParsePtr == NULL)
                    savemalloc = ParsePtr = (char *)malloc(0xfff);
                  if (fread(ParsePtr, tmps+4,   1,  fp) != 0)
                   HwLen = FrameLen = (tmps+4) * 8;        // lanz issue, or work w/ .pfds
                  soughtindex++;
                  }
```

```
                }
              else
                {
                OkMessage (hwnd, "Could not read more frames %s!", TraceTitleName) ;
                szFileName [0] = TraceTitleName [0] = '\0' ;
                fclose(fp);
                return(0);
                }
              fclose(fp);
              ParseLen = padlines = datalines = 0;
              }
            hwndChild1 = LOWORD (SendMessage (hwndClient,
                        WM_MDIGETACTIVE, 0, 0L)) ;
            InvalidateRect (hwndChild1, NULL, TRUE) ;
            }
          else
            {
            OkMessage (hwnd, "Analyzer has not been RUN yet %s!", TraceTitleName) ;
            szFileName [0] = TraceTitleName [0] = '\0' ;
            }
          return (0);

        case IDM_PREVFRAME:

          if (aptl == 0)
            {
            OkMessage (hwnd, "Could not read protocol template file %s!", TraceTitleName) ;
            szFileName [0] = TraceTitleName [0] = '\0' ;
            return(0);
            }
          else if (soughtindex > 1)
            {
            sought=0;
            FILE *fp = fopen(TraceFileName, "rb");
            U16 tmps;
            if (fp != 0)
            for (U32 i=0; i<100; i++)
```

```
                {
                int qq = fseek(fp, sought, SEEK_SET);   // length field
                if (qq==0)
                  {
                  fread(&tmps,   sizeof(tmps),   1,  fp);
                  if (tmps != 0x1005)
                    {
                    fread(&tmps,   sizeof(tmps),   1,  fp);
                    if (tmps < 0xfff)            // some #
                      sought +=(tmps+4);     // position of next length field
                    else
                      {
                      fclose(fp);
                      OkMessage (hwnd, "Could not read more frames %s!", TraceTitleName)
;
                      szFileName [0] = TraceTitleName [0] = '\0' ;
                      return(0); // give some message
                      }
                    }
                  }
                else break;
                }
    int qq;
               for (U32 i = 0; i < (soughtindex-1); i++)
                  {
                  qq = fseek(fp, sought+2, SEEK_SET);   // length field
                  fread(&tmps,   sizeof(tmps),   1,  fp);
                  if (tmps < 0xfff)            // some #
                    sought +=(tmps+4);     // position of next length field
                  else
                    {
                    fclose(fp);
                    OkMessage (hwnd, "Could not read more frames %s!", TraceTitleName) ;
                    szFileName [0] = TraceTitleName [0] = '\0' ;
                    return(0); // give some message
                    }
                  }
```

```
                qq = fseek(fp, -4, SEEK_CUR);   // start of frame
                if (ParsePtr == NULL)
                  savemalloc = ParsePtr = (char *)malloc(0xfff);
                if (fread(ParsePtr, tmps+4,  1,  fp) != 0)
                  {
                  HwLen = FrameLen = (tmps+4) * 8;        // lanz issue, or work w/ .pfds
                  fclose(fp);
                  ParseLen = padlines = datalines = 0;
                  soughtindex--;
                  sought+=2;
                  hwndChild1 = LOWORD (SendMessage (hwndClient,
                                WM_MDIGETACTIVE, 0, 0L));
                  InvalidateRect (hwndChild1, NULL, TRUE);

                  }
                }
              return(0);

            case IDM_ALYZERCLOSE:

              TraceFileName [0] = TraceTitleName [0] = '\0';
              hwndChild1 = LOWORD (SendMessage (hwndClient,
                            WM_MDIGETACTIVE, 0, 0L));

    //          if (SendMessage (hwndChild1, WM_QUERYENDSESSION, 0, 0L))
                SendMessage (hwndClient, WM_MDIDESTROY,
                        hwndChild1, 0L);
            if (savemalloc != NULL)
              {
              free(savemalloc);
              savemalloc=ParsePtr=NULL;
              }
            return(0);

            default:        // Pass to active child
```

```
                hwndChild1 = LOWORD (SendMessage (hwndClient,
                                WM_MDIGETACTIVE, 0, 0L)) ;

                if (IsWindow (hwndChild1))
                   SendMessage (hwndChild1, WM_COMMAND,
                                wParam, lParam) ;

                break ;      // and then to DefFrameProc
              }
            break ;

          case WM_QUERYENDSESSION:
          case WM_CLOSE:              // Attempt to close all children

            SendMessage (hwnd, WM_COMMAND, IDM_CLOSEALL, 0L) ;

            if (NULL != GetWindow (hwndClient, GW_CHILD))
               return 0 ;

            break ;  // ie, call DefFrameProc ;

          case WM_DESTROY :
            PostQuitMessage (0) ;
            return 0 ;
          }
            // Pass unprocessed messages to DefFrameProc (not DefWindowProc)

        return DefFrameProc (hwnd, hwndClient, message, wParam, lParam) ;
        }

      BOOL FAR PASCAL _export CloseEnumProc (HWND hwnd, LONG lParam)
        {
        if (GetWindow (hwnd, GW_OWNER))      // check for icon title
          return 1 ;

        SendMessage (GetParent (hwnd), WM_MDIRESTORE, hwnd, 0L) ;
```

```
    if (!SendMessage (hwnd, WM_QUERYENDSESSION, 0, 0L))
        return 1 ;

    SendMessage (GetParent (hwnd), WM_MDIDESTROY, hwnd, 0L) ;
    lParam = 0;       // To eliminate compiler warning !!!
    return 1 ;
    }


long FAR PASCAL _export DisplayImageWndProc (HWND hwnd, UINT message, UINT wParam,
                                             LONG lParam)
{
CLIENTCREATESTRUCT clientcreate ;
static HWND     hwndClient; //, hwndFrame ;
PAINTSTRUCT     ps ;
TEXTMETRIC      tm ;
U32 x = 0, y = 0;
protocol  *ptl=0;

ptl = (protocol *)GetWindowLong(hwnd, 0);

switch (message)
  {
  case WM_CREATE:

      lParam = ((MDICREATESTRUCT far *)MK_FP32(((CREATESTRUCT far *)MK_FP32((void
*)lParam))->lpCreateParams))->lParam;
    SetWindowLong (hwnd, 0, lParam) ;
        clientcreate.hWindowMenu  = hMenuWindow ;
        clientcreate.idFirstChild = IDM_FIRSTCHILD ;
        hwndClient = CreateWindow ("MDICLIENT", NULL,
                          WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE | WS_VSCROLL |
WS_HSCROLL,
                  0, 0, 0, 0, hwnd, 1, hInst,
                  (LPSTR) &clientcreate) ;
        savepkt = ParsePtr; // = pkt2;  //pkt1;
    return 0 ;
```

```
        case WM_MOVE:
          ptl->setupOutlineCoordsMult(0);
          InvalidateRect (hwnd, NULL, TRUE) ;
          return 0 ;


        case WM_SIZE:

          hdc = GetDC (hwnd) ;
          GetTextMetrics (hdc, &tm) ;
          cyChar = tm.tmHeight + tm.tmExternalLeading ;
          cxChar = tm.tmAveCharWidth;
          ReleaseDC (hwnd, hdc) ;
          cxClient = LOWORD (lParam);
          cyClient = HIWORD (lParam);
          ptl->setupOutlineCoordsMult(0);
          InvalidateRect (hwnd, NULL, TRUE) ;
    //      return(0);


           nVscrollMax = max (0, NUMLINES + 2 - cyClient / cyChar) ;
             nVscrollPos = min (nVscrollPos, nVscrollMax) ;

             SetScrollRange (hwnd, SB_VERT, 0, (WORD)nVscrollMax, FALSE) ;
             SetScrollPos   (hwnd, SB_VERT, (WORD)nVscrollPos, TRUE) ;

           nHscrollMax = max (0, 2 + (nMaxWidth - cxClient) / cxChar) ;
             nHscrollPos = min (nHscrollPos, nHscrollMax) ;

             SetScrollRange (hwnd, SB_HORZ, 0, (WORD)nHscrollMax, FALSE) ;
             SetScrollPos   (hwnd, SB_HORZ, (WORD)nHscrollPos, TRUE) ;
             return 0 ;


           case WM_VSCROLL:
             switch (wParam)
                {
```

```
            case SB_TOP:
               nVscrollInc = -nVscrollPos ;
               break ;

            case SB_BOTTOM:
               nVscrollInc = nVscrollMax - nVscrollPos ;
               break ;

            case SB_LINEUP:
               nVscrollInc = -1 ;
               break ;

            case SB_LINEDOWN:
               nVscrollInc = 1 ;
               break ;

            case SB_PAGEUP:
               nVscrollInc = min (-1, -cyClient / cyChar) ;
               break ;

            case SB_PAGEDOWN:
               nVscrollInc = max (1, cyClient / cyChar) ;
               break ;

            case SB_THUMBTRACK:
               nVscrollInc = LOWORD (lParam) - nVscrollPos ;
               break ;

            default:
               nVscrollInc = 0 ;
            }
         nVscrollInc = max (-nVscrollPos,
                  min (nVscrollInc, nVscrollMax - nVscrollPos)) ;

         if (nVscrollInc != 0)
            {
            nVscrollPos += nVscrollInc ;
```

```
            ScrollWindow (hwnd, 0, WORD(-cyChar * nVscrollInc), NULL, NULL) ;
            SetScrollPos (hwnd, SB_VERT, (WORD)nVscrollPos, TRUE) ;
        InvalidateRect (hwnd, NULL, TRUE) ;
    DestroyWindow(hDlgModeless);
    hDlgModeless    = CreateDialogParam(hInst, szAnlzClass, hwnd,
                            MakeProcInstance ((FARPROC) AnlzWndProc, hInst), 0);

    SendMessage (hDlgModeless, WM_INITDIALOG, 0, 0L) ;


          }
        return 0 ;

    case WM_COMMAND:

      return 0 ;

    case WM_PAINT: // Paint the window

      hdc = BeginPaint (hwnd, &ps) ;
      nPaintBeg = ps.rcPaint.top;
      nPaintEnd = ps.rcPaint.bottom;
      ptl->setupOutlineCoordsMult(0);
      while(ptl != 0)
        {
        ptl = ParseProtocol2(ptl);
        if (ptl!=0) ptl->setupOutlineCoordsMult(TRUE);
        }
      ParsePtr=savepkt;
      ParseLen = padlines = datalines = 0;
      EndPaint (hwnd, &ps) ;
      return 0 ;

    case WM_MDIACTIVATE:

      return 0 ;
```

```
    case WM_QUERYENDSESSION:
    case WM_CLOSE:

      if (IDOK != MessageBox (hwnd, "OK to close window?", "",
                              MB_ICONQUESTION | MB_OKCANCEL))
        return 0 ;
      break ;  // ie, call DefMDIChildProc

    case WM_DESTROY:
      return 0 ;

    case WM_LBUTTONDOWN:
      x = LOWORD(lParam);
      y = HIWORD(lParam);
      ptl-> FindField(hwnd, x, y);
      SetFocus(GetWindow(hwnd, GW_CHILD));
      return 0 ;

    } // Pass unprocessed message to DefMDIChildProc
  return(DefMDIChildProc (hwnd, message, wParam, lParam));
  }

  long FAR PASCAL _export ImageWndProc (HWND hwnd, UINT message, UINT wParam,
                                        LONG lParam)
  {
  static HWND    hwndClient; //, hwndFrame ;
  HDC          hdc ;
  PAINTSTRUCT    ps ;
  TEXTMETRIC     tm ;
  U32 x = 0, y = 0;
  protocol  *ptl=0;

  ptl = (protocol *)GetWindowLong(hwnd, 0);
  switch (message)
    {
    case WM_CREATE:
```

```
// lParam is ptr to CREATESTRUCT; lpCreateParams is ptr to MDICREATESTRUCT; lParam is from
WM_MDICREATE msg
        lParam = ((MDICREATESTRUCT far *)MK_FP32(((CREATESTRUCT far *)MK_FP32((void
*)lParam))->lpCreateParams))->lParam;
      SetWindowLong (hwnd, 0, lParam);
      hwndClient = GetParent (hwnd);       // Save CLIENT window handle
//     hwndFrame  = GetParent (hwndClient); // Save FRAME window handle
      return 0;

    case WM_MOVE:
      ptl->setupOutlineCoordsMult(0);
      InvalidateRect (hwnd, NULL, TRUE);
      return 0;

    case WM_SIZE:

      hdc = GetDC (hwnd);
      GetTextMetrics (hdc, &tm);
      cyChar = tm.tmHeight + tm.tmExternalLeading;
      cxChar = tm.tmAveCharWidth;
      ReleaseDC (hwnd, hdc);
      cxClient = LOWORD (lParam);
      cyClient = HIWORD (lParam);
      ptl->setupOutlineCoords();
      InvalidateRect (hwnd, NULL, TRUE);
      return(0);

    case WM_COMMAND:

      return 0;

    case WM_PAINT: // Paint the window

      hdc = BeginPaint (hwnd, &ps);
      nPaintBeg = ps.rcPaint.top;
      nPaintEnd = ps.rcPaint.bottom;
      if (ptl != 0)
```

```
            ptl->OutlineProtocol(hdc);
          EndPaint (hwnd, &ps) ;
          return 0 ;

        case WM_MDIACTIVATE:

          return 0 ;

        case WM_QUERYENDSESSION:
        case WM_CLOSE:

          if (IDOK != MessageBox (hwnd, "OK to close window?", "",
                                  MB_ICONQUESTION | MB_OKCANCEL))
            return 0 ;
          break ;  // ie, call DefMDIChildProc

        case WM_DESTROY:
          return 0 ;

        case WM_LBUTTONDOWN:
          x = LOWORD(lParam);
          y = HIWORD(lParam);
          if (ptl != 0)
            ptl->FindField(hwnd, x, y);
          SetFocus(GetWindow(hwnd, GW_CHILD));
          return 0 ;

        } // Pass unprocessed message to DefMDIChildProc
      return(DefMDIChildProc (hwnd, message, wParam, lParam));
      }
      void InsertComboBoxNumbers(HWND hwnd, WORD ControlId, WORD Default, U32 minimum, U32
      maximum, U32 increment)
      {
      U32 i, idx, def=-1;
      char buffer[80];
```

```
    hwnd = GetDlgItem(hwnd, ControlId);
    SendMessage(hwnd, CB_RESETCONTENT, 0, 0);
    for (i=minimum; i<=maximum; i+=increment)
      {
      wsprintf(buffer, "%u", i);
      idx = SendMessage(hwnd, CB_ADDSTRING, 0, (LPARAM)((LPSTR)buffer));
      if (i == Default)
        def = idx;
      }
    if (def != -1)
      SendMessage(hwnd, CB_SETCURSEL, (WORD)def, 0);
    }
    void SetupPtlParameters(HWND hwnd, protocol *ptl)
    {
    SendDlgItemMessage(hwnd, PTL_OPTSAVAIL,  CB_RESETCONTENT,  0,  0);
    SetDlgItemInt(hwnd, PTL_BITLEN,  (WORD)ptl->numbits(), FALSE);
    SetDlgItemInt(hwnd, PTL_NUMFIELDS, (WORD)ptl->numfields(), FALSE);
    SetDlgItemText(hwnd, PTL_NAME,  (LPCSTR )ptl->pname());
    protocol *popts=ptl->options();
    if ( popts != 0)
      SetDlgItemText(hwnd, PTL_OPTSAVAIL,  (LPCSTR )popts->pname());
    else  SetDlgItemText(hwnd, PTL_OPTSAVAIL,  (LPCSTR)("[None]" ));
    SendDlgItemMessage(hwnd, PTL_OPTSAVAIL, CB_INSERTSTRING,
                (WPARAM)0, (LPARAM)(LPSTR)("[None]"));

    for (U32 i=1; i<ProtocolList->entries(); i++)
      SendDlgItemMessage(hwnd, PTL_OPTSAVAIL, CB_INSERTSTRING,
                (WPARAM)i, (LPARAM)(LPSTR)ProtocolList->operator[](i).prot()->pname());
    }
    void SetupFieldParameters(HWND hwnd, protocol *ptl)
    {
    static HWND hwndTmp=0;
    static HWND hwndTmp2=0;
    field *fld = ptl->fieldptr(ptl->curfield());
    //    Edit Boxes
    SetDlgItemInt (hwnd, FIELD_NUMBER, (UINT)(ptl->curfield()+1), FALSE);
    SetDlgItemText(hwnd, FIELD_NAME,  (LPCSTR)fld->name());
```

```
    //    Combo Boxes
    SetDlgItemInt(hwnd, FIELD_OFFSET,    (WORD)fld->bitoffset(),   FALSE);
    SetDlgItemInt(hwnd, FIELD_LEN,       (WORD)fld->bitlen(),      FALSE);
    //InsertComboBoxNumbers(hwnd, FIELD_NEXT,   (WORD)(fld->curfield()+1),     1, 256, 1);
    //    Check Boxes
    CheckDlgButton(hwnd, FIELD_NEXTPTL,  (WORD)(fld->lookptr() != 0));
    CheckDlgButton(hwnd, FIELD_PKTLEN,   (WORD)(fld->protlen() != 0));
    CheckDlgButton(hwnd, FIELD_HDRLEN,   (WORD)(fld->hdlen()   != 0));
    CheckDlgButton(hwnd, FIELD_BYTESWAP, (WORD)(fld->swap() != 0));
    //CheckDlgButton(hwnd, FIELD_FILTER,   (WORD)(fld->critptr()  != 0));
    CheckDlgButton(hwnd, FIELD_COUNTBITS, (WORD)(fld->bitlen()   == 0));
    if (hwndTmp != NULL)
      {
      DestroyWindow(hwndTmp);
      hwndTmp = 0;
      }
    if (hwndTmp2 != NULL)
      {
      DestroyWindow(hwndTmp2);
      hwndTmp2 = 0;
      }

    if (fld->lookptr())
      {
      hwndTmp = CreateDialogParam(hInst, szNxtPtlClass, GetParent(hwnd),
                        MakeProcInstance ((FARPROC) NxtPtlWndProc, hInst), (LPARAM)ptl);
      SetFocus(hwndTmp);
      }

    //if (fld->critptr())
    //  {
    //  hwndTmp2 = CreateDialogParam(hInst, szFilterClass, GetParent(hwnd),
    //                    MakeProcInstance ((FARPROC) FilterWndProc, hInst), (LPARAM)ptl);
    //  SetFocus(hwndTmp2);
    //  }
    }
```

```
field f;

long FAR PASCAL _export ConfigWndProc (HWND hwnd, UINT message, UINT wParam, LONG lParam)
{
HWND            hwndParent;
BOOL            err;
U32             tmp, len, off, actualoff, ii;
static S32      newbits;
char            bufr[50];
static protocol *ptl=0;
static field    *fld=0;

field *f, *g;       // for searching for duplicate/overlayed fields FIELD_OFFSET

//
// Get handle for Parent window
// Get pointer to protocol associated with Parent window
// Set up current field pointer
//
hwndParent = GetParent(hwnd);
ptl = (protocol *)GetWindowLong(hwndParent, 0);
fld = ptl->fieldptr(ptl->curfield());
protocols p(ptl->pname(), ptl);
switch (message)
  {
  case WM_ACTIVATE:
    if (wParam == 0)  // Window has been deactivated !!!
      return(FALSE);

  case WM_SETFOCUS:
  case WM_INITDIALOG:
//  Set up Protocol Specific Parameters
    SetDlgItemInt(hwnd, PTL_BITLEN,   (WORD)ptl->numbits(), FALSE);
    SetDlgItemInt(hwnd, PTL_NUMFIELDS, (WORD)ptl->numfields(), FALSE);
    InsertComboBoxNumbers(hwnd, PTL_DSPWIDTH, (WORD)ptl->dspbW(),     8, 112, 8);
//  Set up Field Specific Parameters
    SetupFieldParameters(hwnd, ptl);
```

```
        SetupPtlParameters(hwnd, ptl);
        return(FALSE);

      case WM_COMMAND:

        switch(wParam)
          {
          case PTL_DSPWIDTH:
            tmp = GetDlgItemInt(hwnd, wParam, &err, FALSE);
            if (err == FALSE || tmp < 8 || tmp > 112)
              return(0);
            ptl->dspbW(tmp);
            ptl->setupOutlineCoords();
            break;

          case PTL_NAME:

            GetDlgItemText(hwnd, wParam, (LPSTR)bufr, sizeof(bufr)-1);
            if (HIWORD(lParam) == 0x100)
              NAMEOK100 = TRUE;
            if (NAMEOK100 == FALSE)   return(0);
            if (HIWORD(lParam) == 0x300)
              {
              for ( ii=0; ii<ProtocolList->entries(); ii++)
                if (strcmp( ptl->pname(),
                  ProtocolList->operator[](ii).prot()->pname()) == 0)
                  {
                  ProtocolList->removeAt(ii);
                  break;
                  }
              ptl->pname(bufr);
              ptl->pnamelen(strlen(bufr)+1);
              ProtocolList->insert(p);
              }
            break;
```

```
        case PTL_OPTSAVAIL:

            U32 idx;
              if ((idx=SendDlgItemMessage(hwnd, PTL_OPTSAVAIL, CB_GETCURSEL, 0, 0)) !=
    CB_ERR)
              {
    // Put selected text into NEXTPTLS edit and list boxes
                SendDlgItemMessage(hwnd, PTL_OPTSAVAIL, CB_GETLBTEXT, (WPARAM)idx,
    (LPARAM)(LPSTR)bufr);
              for (U32 i=0; i<ProtocolList->entries(); i++)
                {
                if (strcmp( ProtocolList->operator[](i).prot()->pname(),
                  bufr) == 0) break;
                }
              if (i!=ProtocolList->entries())
                ptl->options(ProtocolList->operator[](i).prot());
              else
                ptl->options(NULL);
              }

          return(FALSE);

        case PTL_BITLEN:
        case PTL_NUMFIELDS:
          break;

        case PTL_ADD:
          if (ptl->numfields() > ptl->curfield())
            ptl->add_field(ptl->curfield(),ptl);
          SetupFieldParameters(hwnd, ptl);
          SetupPtlParameters(hwnd, ptl);
          ptl->setupOutlineCoords();
          InvalidateRect (GetParent(hwnd), NULL, TRUE);
    // NEED STUFF FOR CHECKING FIELD BIT OFFSET, NOT ADDING LENGTH IF OVERLAP....
          break;

        case PTL_AFTER :
```

```
            if (ptl->numfields() > ptl->curfield())
              ptl->add_field_after(ptl->curfield(),ptl);
            SetupFieldParameters(hwnd, ptl);
            SetupPtlParameters(hwnd, ptl);
            ptl->setupOutlineCoords();
            InvalidateRect (GetParent(hwnd), NULL, TRUE);
        // NEED STUFF FOR CHECKING FIELD BIT OFFSET, NOT ADDING LENGTH IF OVERLAP..
            break;

          case PTL_DELETE:
            if (ptl->numfields() > 1)
              ptl->delete_field(ptl->curfield(),ptl);
            SetupFieldParameters(hwnd, ptl);
            SetupPtlParameters(hwnd, ptl);
            break;

          case FIELD_NAME:
            if (GetDlgItemText(hwnd, wParam, (LPSTR)bufr, sizeof(bufr)-1) != 0)
              fld->name(bufr);
            break;

          case FIELD_COUNTBITS:
            switch(IsDlgButtonChecked(hwnd, wParam))
              {
              case 1:                  // don't count bits in total bit len
                fld->bitlen(0);
                break;
              case 0:                  // use bit len in offset/len calcs
                SendMessage(hwnd, WM_COMMAND, FIELD_LEN,
                  (LPARAM)(lParam=CBN_SELENDOK<<16));
                break;
              }


          case FIELD_OFFSET:
          case FIELD_LEN:
            if (wParam == FIELD_OFFSET)
```

```
                {
                if (HIWORD(lParam) == 0x100)
                  OFFSETOK100 = TRUE;
                if (OFFSETOK100 == FALSE)       return(0);
                if (HIWORD(lParam) != 0x300) return(0);
                }
              else if (wParam == FIELD_LEN)
                {
                if (HIWORD(lParam) == 0x100)
                  BITLENOK100 = TRUE;
                if (BITLENOK100 == FALSE)   return(0);
                if (HIWORD(lParam) != 0x300) return(0);
                }
              off = GetDlgItemInt(hwnd, FIELD_OFFSET, &err, FALSE);
              if (err != FALSE)
                {
                len = GetDlgItemInt(hwnd, FIELD_LEN, &err, FALSE);
                if (err != FALSE)
                  {
                  newbits = len - fld->bitlen();
    //
    // Case where the field is not word aligned
    //
                U32 xx;
                if ( ((xx=(off & 31)) !=0) && (len > (32 - xx)) )
                  actualoff = off & (~7);
                else
                  actualoff = off & (~31);
                  fld->offset((U16)(actualoff/8));
                  fld->shlbits((U8)(off - actualoff));
                  fld->shrbits((U8)(32 - len));
                  ptl->numbits(ptl->numbits()- fld->bitlen()+len);
                  fld->bitlen(len);
    //
    // Check for duplicate fields - get new total bitlen
    //
                  f = g = ptl->fieldptr();
```

```
//              ptl->numbits(0);
                for (ii=0; ii<ptl->numfields(); ii++)
                  {
                  f = &g[ii];
                  if (ii!=ptl->curfield())
                    {
//
// Adjust position of following fields
//
                      if (fld->bitoffset() < f->bitoffset())
                        {
                        off = f->bitoffset() + newbits;
                        len = f->bitlen();
                        actualoff = off & (~31);
                        f->offset((U16)(actualoff/8));
                        f->shlbits((U8)(off - actualoff));
                        f->shrbits((U8)(32 - len));
                        }
                      }
                    }
                  }
                SetupPtlParameters(hwnd, ptl);
                break;

              case FIELD_NUMBER:
                break;

              case FIELD_PREVIOUS_BUTTON:

                if (ptl->prevfield() == FALSE)
                  return(0);
                SetupFieldParameters(hwnd, ptl);
                break;

              case FIELD_NEXT_BUTTON:
```

```
            if (ptl->nextfield() == FALSE)
              return(0);
            SetupFieldParameters(hwnd, ptl);
            break;

        case FIELD_NEXTPTL:

            switch(IsDlgButtonChecked(hwnd, wParam))
              {
              case 0:
                if (fld->lookptr()!=0)
                  fld->lookptr()->~lookup();
                fld->lookptr(NULL);
    //  ~verify(); for array & onevalue: need to ~verify
                break;

              case 1:
                if (fld->lookptr() == NULL)
                  fld->lookptr(alloc_lookup_structs(ONEVALUE, fld));  // something better than this?
              break;

              }
            SetupFieldParameters(hwnd, ptl);
            break;

    //
    /*      case FIELD_FILTER:

            switch(IsDlgButtonChecked(hwnd, wParam))
              {
              case 0:
                if (fld->critptr()!=0)
                  fld->critptr()->~criteria();
                fld->critptr(NULL);
    // any need for? ~verify(); for array & onevalue: need to ~verify
    // delete[]???
                break;
```

```
                case 1:
                  if (fld->critptr() == NULL)
                    {
                    criteria *c = new criteria;
                    fld->critptr(c);
                    fld->critptr()->FillInCriteria(0, NULL, NULL, ptl->pname(),
                      fld->name());
                    fld->critptr()->critrange(alloc_lookup_tree());
                    }
                  break;

                  }
                SetupFieldParameters(hwnd, ptl);
                break;
    */

    //
            case FIELD_PKTLEN:

              CheckDlgButton(hwnd, wParam, (WORD)fld->protlen(!fld->protlen()));
              break;

            case FIELD_BYTESWAP:

              fld->swap(!fld->swap());
              CheckDlgButton(hwnd, wParam, fld->swap());
              break;

            case FIELD_HDRLEN:

    //          CheckDlgButton(hwnd, wParam, (WORD)fld->hdlen(!fld->hdlen()));
    // CHANGE TYPE IN DECODE.HPP...?
              CheckDlgButton(hwnd, wParam, (WORD)(IsDlgButtonChecked(hwnd, wParam) ? 0:1));
              fld->hdlen(IsDlgButtonChecked(hwnd, wParam) ? 0:1);
              SetupFieldParameters(hwnd, ptl);
              break;
```

```
            default:
               return(0);
            }
    //      r.top = (WORD)nPaintBeg; r.bottom = (WORD)yB; r.left = (WORD)xL; r.right = (WORD)xR;
            InvalidateRect(hwndParent, NULL, TRUE); // Invalidate entire Image Window rectangle (Our
    Parent)
          return(0);

        case WM_SIZE:
        case WM_MOVE:

            InvalidateRect(hwndParent, NULL, TRUE); // Invalidate entire Image Window rectangle (Our
    Parent)
          return(0);

        case WM_CLOSE:

          InvalidateRect(hwndParent, NULL, TRUE);
          SendMessage(hwndParent, WM_CLOSE, 0, 0);
          DestroyWindow(hwnd);
          return(0);
        }
    SetActiveWindow(hwnd);
    lParam = 0; // Make compiler complaint go away
    return(FALSE);
    }

    long FAR PASCAL _export AnlzWndProc (HWND hwnd, UINT message, UINT wParam, LONG lParam)
    {
    static HWND       hwndParent, hwndFrame, hwndTmp;
    BOOL              err;

    hwndParent = GetParent(hwnd);
    hwndFrame  = GetParent (hwndParent) ;  // Save FRAME window handle
    hwndTmp = GetParent(hwndFrame);
    switch (message)
      {
```

```
        case WM_ACTIVATE:
          if (wParam == 0)  // Window has been deactivated !!!
            return(FALSE);

        case WM_SETFOCUS:
        case WM_INITDIALOG:
          SetDlgItemInt(hwnd, ANLZ_CUR,   (WORD)soughtindex, FALSE);
          SetDlgItemInt(hwnd, ANLZ_TOTAL, (WORD)total, FALSE);
          ShowWindow(hDlgModeless, SW_SHOW);
          return(FALSE);
    //Setup ??

        case WM_COMMAND:

          switch(wParam)
            {
            case ANLZ_NEXT:
              SendMessage(hwndTmp,WM_COMMAND, IDM_NEXTFRAME,0);
              SetDlgItemInt(hwnd, ANLZ_CUR,   (WORD)soughtindex, FALSE);
              break;

            case ANLZ_PREV:
              SendMessage(hwndTmp,WM_COMMAND, IDM_PREVFRAME,0);
              SetDlgItemInt(hwnd, ANLZ_CUR,   (WORD)soughtindex, FALSE);
              break;

            case ANLZ_CUR:
              if (HIWORD(lParam) == 0x100)
                ANLZ_OK100 = TRUE;
              if (ANLZ_OK100 == FALSE)  return(0);
              if (HIWORD(lParam) == 0x300)
                {
                soughtindex = GetDlgItemInt(hwnd, ANLZ_CUR, &err, FALSE);
                if (err != FALSE)
                  {
                  soughtindex++;
                  SendMessage(hwndTmp, WM_COMMAND, IDM_PREVFRAME, 0L );
```

```
                }
    ////?? set ANLZ_OK100 back to FALSE
                }
              break;

        case ANLZ_RUN:
          SendMessage(hwndTmp, WM_COMMAND, IDM_RUN, 0L );
          break;

        case ANLZ_QUIT:
          SendMessage(hwndTmp, WM_COMMAND, IDM_CLOSE, 0L );
          break;

        default:
          break;

        }


    case WM_SIZE:
    case WM_MOVE:

      InvalidateRect(hwndParent, NULL, TRUE); // Invalidate entire Image Window rectangle (Our
  Parent)
      return(0);


    case WM_CLOSE:

      InvalidateRect(hwndParent, NULL, TRUE);
      SendMessage(hwndParent, WM_CLOSE, 0, 0);
      DestroyWindow(hwnd);
      return(0);
    }


  SetActiveWindow(hwnd);
```

```
return(FALSE);

}

////////////////////////////////////////////////////////////////////////
//    NXTPTL.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"

void SetupNxtPtlFields(HWND hwnd, field *fld)
{
verify *vptr;
U32 i, num, curidx=0; // Get from field class member eventually ???
char buffer[80];

CheckRadioButton(hwnd, NP_SIGNED,  NP_UNSIGNED, NP_UNSIGNED);          // defaulting to
unsigned for the moment
CheckRadioButton(hwnd, NP_DECIMAL, NP_HEX, (WORD)(fld->format()==1 ? NP_DECIMAL :
NP_HEX));
CheckRadioButton(hwnd, NP_ODD, NP_ALL, (WORD)(NP_ALL));
CheckDlgButton(hwnd, (WORD)(NP_ONEVALUE-1+fld->lookptr()->get_type()), TRUE);

SendDlgItemMessage(hwnd, NP_MINLIST,     LB_RESETCONTENT,     0,    0);
SendDlgItemMessage(hwnd, NP_MAXLIST,     LB_RESETCONTENT,     0,    0);
SendDlgItemMessage(hwnd, NP_NEXTIDXLIST, LB_RESETCONTENT,     0,    0);
SendDlgItemMessage(hwnd, NP_NEXTPTLLIST, LB_RESETCONTENT,     0,    0);
SendDlgItemMessage(hwnd, NP_NEXTPTLLIST, LB_SETHORIZONTALEXTENT, 200, 0);
SendDlgItemMessage(hwnd, NP_PTLS,        LB_RESETCONTENT,     0,    0);
SendDlgItemMessage(hwnd, NP_MINEDIT,     EM_SETSEL,           0,    MAKELONG(0,-1));
SendDlgItemMessage(hwnd, NP_MINEDIT,     WM_CLEAR,            0,    0);
SendDlgItemMessage(hwnd, NP_MAXEDIT,     EM_SETSEL,           0,    MAKELONG(0,-1));
SendDlgItemMessage(hwnd, NP_MAXEDIT,     WM_CLEAR,            0,    0);
SendDlgItemMessage(hwnd, NP_NEXTIDXEDIT, EM_SETSEL,           0,    MAKELONG(0,-1));
SendDlgItemMessage(hwnd, NP_NEXTIDXEDIT, WM_CLEAR,            0,    0);
SendDlgItemMessage(hwnd, NP_NEXTPTLEDIT, EM_SETSEL,           0,    MAKELONG(0,-1));
SendDlgItemMessage(hwnd, NP_NEXTPTLEDIT, WM_CLEAR,            0,    0);
```

```
if (fld->lookptr() == NULL) return;
if ((num = fld->lookptr()->entries()) !=0)
  {
  for (i=0; i<num; i++)
    {
    vptr = fld->lookptr()->find_index(i);
    wsprintf(buffer, (fld->format()==1) ? "%u" : "0x%x", vptr->minval);
        SendDlgItemMessage(hwnd, NP_MINLIST, LB_INSERTSTRING, (WPARAM)i,
(LPARAM)(LPSTR)buffer);
    if (i == curidx) SetDlgItemText(hwnd, NP_MINEDIT, (LPCSTR)buffer);
    wsprintf(buffer, (fld->format()==1) ? "%u" : "0x%x", vptr->maxval);
    if (i == curidx) SetDlgItemText(hwnd, NP_MAXEDIT, (LPCSTR)buffer);
        SendDlgItemMessage(hwnd, NP_MAXLIST, LB_INSERTSTRING, (WPARAM)i,
(LPARAM)(LPSTR)buffer);
        if (i == curidx) CheckRadioButton(hwnd, NP_ODD, NP_ALL,
(WORD)(NP_ODD+vptr->okbits-1)); // Controls must be in ODD, EVEN, BOTH order
    wsprintf(buffer, "%u", vptr->nxtidx+1);
    if (i == curidx) SetDlgItemText(hwnd, NP_NEXTIDXEDIT, (LPCSTR)buffer);
        SendDlgItemMessage(hwnd, NP_NEXTIDXLIST, LB_INSERTSTRING, (WPARAM)i,
(LPARAM)(LPSTR)buffer);
    wsprintf(buffer, "%s", (vptr->prot == 0) ? "[None]" : vptr->prot->pname());
    if (i == curidx) SetDlgItemText(hwnd, NP_NEXTPTLEDIT, (LPCSTR)buffer);
        SendDlgItemMessage(hwnd, NP_NEXTPTLLIST, LB_INSERTSTRING, (WPARAM)i,
(LPARAM)(LPSTR)buffer);
    }
//SendDlgItemMessage(hwnd, NP_MINEDIT, EM_SETMODIFY, 0, 0);
SendDlgItemMessage(hwnd, NP_MINLIST, LB_SETCURSEL, 0, 0);
SendDlgItemMessage(hwnd, NP_MAXLIST, LB_SETCURSEL, 0, 0);
SendDlgItemMessage(hwnd, NP_NEXTIDXLIST, LB_SETCURSEL, 0, 0);
SendDlgItemMessage(hwnd, NP_NEXTPTLLIST, LB_SETCURSEL, 0, 0);
  }
for (i=0; i<ProtocolList->entries(); i++)
  SendDlgItemMessage(hwnd, NP_PTLS, LB_INSERTSTRING,
        (WPARAM)i, (LPARAM)(LPSTR)ProtocolList->operator[](i).prot()->pname());


//SendDlgItemMessage(hwnd, NP_PTLS, LB_DIR, 0x37, (LONG )(LPSTR)"*.pfd");
```

```
}
#ifdef notdef
void InsertNewRange(HWND hwnd, U32 &rangeIdx, U32 editId, U32 listId)
{
S8 cbString[50];
      SendDlgItemMessage(hwnd, (WORD)editId, EM_SETSEL, 0, MAKELONG(0,-1));
      GetDlgItemText   (hwnd, (WORD)editId, (LPSTR)cbString, sizeof(cbString));
rangeIdx = SendDlgItemMessage(hwnd, (WORD)listId, LB_INSERTSTRING, (WPARAM)rangeIdx,
(LPARAM)(LPSTR)cbString);
      SendDlgItemMessage(hwnd, (WORD)listId, LB_SETCURSEL, (WPARAM)rangeIdx, 0);
}
#endif

long FAR PASCAL _export NxtPtlWndProc (HWND hwnd, UINT message, UINT wParam, LONG lParam)
{
static U32      rangeIdx=0;
HWND            hwndParent;
BOOL            err;
U32             tmp;
static protocol  *ptl=0;
static field     *fld=0;
static verify    *vptr=0;
char            buffer[80];
static verify tvptr;
switch (message)
  {
  case WM_ACTIVATE:
    if (wParam == 0)  // Window has been deactivated !!!
      return(FALSE);

  case WM_INITDIALOG:
    ptl = (protocol *)lParam;          // Setup ptl each time Dialog is created

  //   Fall Thru
  case WM_SETFOCUS:
    if (fld->lookptr() != NULL)
      fld = ptl->fieldptr(ptl->curfield());  // Recompute fld when acquire input focus
```

```
        vptr = fld->lookptr()->find_index(rangeIdx);
        SetupNxtPtlFields(hwnd, fld);     // Setup each control
        return(FALSE);

      case WM_COMMAND:

        switch(wParam)
          {

          case NP_ONEVALUE:          // 314
          case NP_ARRAY:             // 315
          case NP_TREE:              // 317
              if (fld->lookptr()!=0)
                {
                fld->lookptr()->~lookup();
                fld->lookptr(NULL);
                }
              fld->lookptr(alloc_lookup_structs(wParam-NP_ONEVALUE+1, fld));
              SetupNxtPtlFields(hwnd, fld);     // Display cleared-out ranges

    //
    // Requires that the ONEVALUE,ARRAY,TREE match the order & spacing of the
    // enum lookuptypes{ NOLOOKUP, ONEVALUE, ARRAY, HASH, TREE }
    //
            return(0);

          case NP_MINLIST:
          case NP_MAXLIST:
          case NP_NEXTIDXLIST:
          case NP_NEXTPTLLIST:

            rangeIdx = SendDlgItemMessage(hwnd, wParam, LB_GETCURSEL, 0, 0);
            vptr = fld->lookptr()->find_index(rangeIdx);
            tvptr.prot = vptr->prot;
            tvptr.nxtidx = vptr->nxtidx;
            tvptr.okbits = vptr->okbits;
            tvptr.minval = vptr->minval;
```

```
            tvptr.maxval = vptr->maxval;
            break;

          case NP_ODD:
          case NP_EVEN:
          case NP_ALL:

            tvptr.okbits = wParam - NP_ODD + 1;
            return (0);

          case NP_DECIMAL:
            fld->format(DEC);
            SetupNxtPtlFields(hwnd, fld);     // Setup each control
            return (0);

          case NP_HEX:
            fld->format(HEX);
            SetupNxtPtlFields(hwnd, fld);     // Setup each control
            return (0);

          case NP_PTLS:
    //        if (HIWORD(lParam) != LBN_DBLCLK)
    //          return(FALSE);
          case NP_USE:

              U32 idx; //, len;
              if ((idx=SendDlgItemMessage(hwnd, NP_PTLS, LB_GETCURSEL, 0, 0)) != LB_ERR)
                {
    // Put selected text into NEXTPTLS edit and list boxes
                        SendDlgItemMessage(hwnd, NP_PTLS, LB_GETTEXT, (WPARAM)idx,
    (LPARAM)(LPSTR)buffer);
    //            if (strlen(buffer)!= 0)
                SetDlgItemText(hwnd, NP_NEXTPTLEDIT, (LPCSTR)buffer);
                  {
                  for (U32 i=0; i<ProtocolList->entries(); i++)
                    {
                    if (strcmp( ProtocolList->operator[](i).prot()->pname(),
```

```
                buffer) == 0) break;
              }
            if (i!=ProtocolList->entries())
              tvptr.prot=ProtocolList->operator[](i).prot();
            else
              tvptr.prot = 0;
          }

        }
      return(FALSE);

    case NP_INSERT:
    case NP_MODIFY:

      if (fld->format() == HEX)
        {
        GetDlgItemText(hwnd, NP_MINEDIT, (LPSTR)buffer, sizeof(buffer)-1);
        tvptr.minval =strtol((char *)buffer,NULL,16);
        GetDlgItemText(hwnd, NP_MAXEDIT, (LPSTR)buffer, sizeof(buffer)-1);
        tmp=strtol((char *)buffer,NULL,16);
        tvptr.maxval = tmp;
        }
      else //  (fld->format() == DEC)
        {
        tmp = GetDlgItemInt(hwnd, NP_MINEDIT, &err, FALSE);
        if (err == FALSE)
          return(0);
        tvptr.minval = tmp;
        tmp = GetDlgItemInt(hwnd, NP_MAXEDIT, &err, FALSE);
        if (err == FALSE)
          return(0);
        tvptr.maxval = tmp;
        }

      tmp = GetDlgItemInt(hwnd, NP_NEXTIDXEDIT, &err, FALSE);
      if (err == FALSE)
        return(0);
```

```
            tvptr.nxtidx = tmp-1;
            GetDlgItemText(hwnd, NP_NEXTPTLEDIT, (LPSTR)buffer, sizeof(buffer)-1);
    // ? do this with find instead?
            if (strlen(buffer)!= 0)
              {
              for (U32 i=0; i<ProtocolList->entries(); i++)
                {
                if (strcmp( ProtocolList->operator[](i).prot()->pname(),
                  buffer) == 0) break;
                }
              if (i!=ProtocolList->entries())
                tvptr.prot=ProtocolList->operator[](i).prot();
              else tvptr.prot = 0;
              }
            if (IsDlgButtonChecked(hwnd, NP_ALL))
              tvptr.okbits = ALLNUMBERS;
            else if (IsDlgButtonChecked(hwnd, NP_ODD))
              tvptr.okbits = ODDNUMBERS;
            else if (IsDlgButtonChecked(hwnd, NP_EVEN))
              tvptr.okbits = EVENNUMBERS;

            if (wParam == NP_MODIFY)
              {
              fld->lookptr()->modify_entry(rangeIdx, tvptr.prot, tvptr.nxtidx,
                tvptr.minval, tvptr.maxval, tvptr.okbits);
              }

            if (wParam == NP_INSERT)
              {
              fld->lookptr()->insert_entry(tvptr.prot, tvptr.nxtidx, tvptr.minval,
                tvptr.maxval, tvptr.okbits);
              }

          SetupNxtPtlFields(hwnd, fld);    // Setup each control
          return(FALSE);
```

```
            case NP_DELETE:
              SendDlgItemMessage(hwnd, NP_MINLIST,    LB_DELETESTRING, (WPARAM)rangeIdx, 0);
              SendDlgItemMessage(hwnd, NP_MAXLIST,    LB_DELETESTRING, (WPARAM)rangeIdx, 0);
              SendDlgItemMessage(hwnd, NP_NEXTIDXLIST, LB_DELETESTRING, (WPARAM)rangeIdx,
 5  0);
              SendDlgItemMessage(hwnd, NP_NEXTPTLLIST, LB_DELETESTRING, (WPARAM)rangeIdx,
    0);
              if (fld->lookptr()->entries() > 0)
                fld->lookptr()->delete_entry(rangeIdx, tvptr.minval);
10            break;

            default:
              return(FALSE);
            }
          SendDlgItemMessage(hwnd, NP_MINLIST,      LB_SETCURSEL, (WPARAM)rangeIdx, 0);
15        SendDlgItemMessage(hwnd, NP_MAXLIST,      LB_SETCURSEL, (WPARAM)rangeIdx, 0);
          SendDlgItemMessage(hwnd, NP_NEXTIDXLIST,  LB_SETCURSEL, (WPARAM)rangeIdx, 0);
          SendDlgItemMessage(hwnd, NP_NEXTPTLLIST,  LB_SETCURSEL, (WPARAM)rangeIdx, 0);
          SendDlgItemMessage(hwnd, NP_MINLIST,      LB_SETTOPINDEX, (WPARAM)rangeIdx, 0);
          SendDlgItemMessage(hwnd, NP_MAXLIST,      LB_SETTOPINDEX, (WPARAM)rangeIdx, 0);
20        SendDlgItemMessage(hwnd, NP_NEXTIDXLIST,  LB_SETTOPINDEX, (WPARAM)rangeIdx, 0);
          SendDlgItemMessage(hwnd, NP_NEXTPTLLIST,  LB_SETTOPINDEX, (WPARAM)rangeIdx, 0);
        SendDlgItemMessage(hwnd, NP_MINEDIT,      EM_SETSEL,      0,    MAKELONG(0,-1));
        SendDlgItemMessage(hwnd, NP_MINEDIT,      WM_CLEAR,       0,    0);
        SendDlgItemMessage(hwnd, NP_MAXEDIT,      EM_SETSEL,      0,    MAKELONG(0,-1));
25      SendDlgItemMessage(hwnd, NP_MAXEDIT,      WM_CLEAR,       0,    0);
        SendDlgItemMessage(hwnd, NP_NEXTIDXEDIT,  EM_SETSEL,      0,    MAKELONG(0,-1));
        SendDlgItemMessage(hwnd, NP_NEXTIDXEDIT,  WM_CLEAR,       0,    0);
        SendDlgItemMessage(hwnd, NP_NEXTPTLEDIT,  EM_SETSEL,      0,    MAKELONG(0,-1));
        SendDlgItemMessage(hwnd, NP_NEXTPTLEDIT,  WM_CLEAR,       0,    0);
30        wsprintf(buffer, (fld->format() == 1) ? "%u" : "0x%x", tvptr.minval);
          SetDlgItemText(hwnd, NP_MINEDIT,  (LPCSTR)buffer);
          wsprintf(buffer, (fld->format() == 1) ? "%u" : "0x%x", tvptr.maxval);
          SetDlgItemText(hwnd, NP_MAXEDIT,  (LPCSTR)buffer);
          wsprintf(buffer, "%u", tvptr.nxtidx+1);
35        SetDlgItemText(hwnd, NP_NEXTIDXEDIT,  (LPCSTR)buffer);
          wsprintf(buffer, "%s", (tvptr.prot == 0) ? "[None]" : tvptr.prot->pname());
```

```
        SetDlgItemText(hwnd, NP_NEXTPTLEDIT,   (LPCSTR)buffer);
        CheckRadioButton(hwnd, NP_ODD, NP_ALL, (WORD)(NP_ODD+tvptr.okbits-1)); // Controls must
    be in ODD, EVEN, BOTH order
        return(FALSE);

    case WM_SIZE:
    // Force size to original (don't allow user to change size of dialog box) ???
    case WM_MOVE:

        return(0);

    case WM_CLOSE:

        hwndParent = GetParent(hwnd);
        InvalidateRect(hwndParent, NULL, TRUE);
        SendMessage(hwndParent, WM_CLOSE, 0, 0);
        DestroyWindow(hwnd);
        return(0);
    }
    lParam = 0; // Make compiler complaint go away
    return(FALSE);
    }

    ////////////////////////////////////////////////////////////////////////
    //    PCOLS.CPP
    ////////////////////////////////////////////////////////////////////////

    #include "gen.hpp"


    static protocols DataPtl, PadPtl;
    protocol *DataPtlPtr, *PadPtlPtr;   // PUT THIS IN setup_protocols()
    WCValSortedVector<protocols> *ProtocolList;          // Pointer to sorted list for lookup values

    protocol *base_protocol(S8 *ProtocolName)
    {
    for (U32 i=0; i<ProtocolList->entries(); i++)
```

```
    if (strcmp(ProtocolName, ProtocolList->operator[](i).prot()->filename()) == 0)
      return(ProtocolList->operator[](i).prot());
  return(0);
  }

  void setup_protocols()
  {
  U32 number=0;
  find_t fileblock;

  //
  // Determine number of protocol definitions that exist by counting files
  //
  if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
    {
    do
      {
      ++number;
    } while(_dos_findnext(&fileblock) == 0);
    }

  //
  // Setup a vector twice the size of the current number with number for the grow size
  //
  ProtocolList = new WCValSortedVector<protocols>(number*2, number);

  //
  // Open each protocol definition file; create a protocol definition in memory; insert it into the vector
  //
  if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
    {
    do
      {
      FILE *fp = fopen(fileblock.name, "rb");
      if ((fp != 0) && (fileblock.size > 56))
        {
        U32 tmp;
```

```
        fread(&tmp,    sizeof(tmp),    1,  fp); // Get length of protocol name
    //
        S8 *pname = (S8 *)malloc(tmp);
        fread(pname,    tmp,          1,  fp); // Get file name
        fclose(fp);
        protocol *proto = new protocol(pname, fileblock.name);
    //
        protocols p(pname, proto);
        free(pname);
        ProtocolList->insert(p);
        }
      } while(_dos_findnext(&fileblock) == 0);
    }

    //
    // ? Do we want to do something about duplicate protocol names here?
    //
    //      for (U32 ii=0, jj = 1; ii<ProtocolList->entries(); ii++, jj++)
    //        {
    //        if ((ii != jj) && (strcmp( ProtocolList->operator[](ii).prot()->pname(),
    //          ProtocolList->operator[](jj).prot()->pname()) == 0))
    //        ProtocolList->removeAt(jj);
    //        }


    //
    // Open each protocol definition file; create all required data structures
    //
    if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
      {
      do
        {
        FILE *fp = fopen(fileblock.name, "rb");
        if ((fp != 0) && (fileblock.size > 56))
          {
          protocols p(fp), t;   // Reads name_length and name
          if (ProtocolList->find(p, t) != FALSE)
```

```
            t.prot()->get_from_file(fp);
          fclose(fp);
          }
        } while(_dos_findnext(&fileblock) == 0);
    }

    protocols data("DATA", 0), pad("PAD", 0);          // PUT THIS IN setup_protocols()
    ProtocolList->find(data, DataPtl);                 // PUT THIS IN setup_protocols()
    ProtocolList->find(pad, PadPtl);                   // PUT THIS IN setup_protocols()
    DataPtlPtr = DataPtl.prot();                       // PUT THIS IN setup_protocols()
    PadPtlPtr = PadPtl.prot();                         // PUT THIS IN setup_protocols()

    //setup_criteria();                 // filter channels are setup - tie criteria to fields
    }

    protocol *setup_newprotocol(S8* szFileName)
    {
    U32 number=0;
    find_t fileblock;
    S8 *pname = (S8 *)malloc(50);
    //
    strcpy(pname,szFileName);
    strcpy(strrchr(pname, '.'), "\0 ");
    strcpy(pname, (strrchr(pname, '\\')+1));
    strcpy(szFileName, (strrchr(szFileName, '\\')+1));
    //
    // See if there are any protocols
    //
    if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
      {
      do
        {
        ++number;
        } while(_dos_findnext(&fileblock) == 0);
      }

    //
```

```
// Setup a vector if there are no .pdfs
//
if (number == 0)
ProtocolList = new WCValSortedVector<protocols>(10, 5);  // use some other define/default

protocol   *proto = new protocol(pname, szFileName);
protocols  p(pname, proto);
field      *tmpfs = new field[1];
field      f(0,0,24);

tmpfs[0] = f;
proto->fieldptr(tmpfs);
proto->numbits(tmpfs->bitlen());
proto->numfields(1);

free(pname);
ProtocolList->insert(p);
FILE *fp = fopen(szFileName, "wb");
if (fp != 0)
   {
   p.prot()->out_to_file();
   fclose(fp);
   p.prot()->clear_out_flag();
   return(proto);
   }
return(FALSE);
}

/*-----------------------------------------
  POPFILE.C -- Popup Editor File Functions
  -----------------------------------------*/

#define INCLUDE_COMMDLG_H    1
#include <windows.h>
#include <stdlib.h>
#include "gen.hpp"
#define  PDF "*.PDF"
```

```
#define TR1 "*.TR1"

static OPENFILENAME ofn ;
//
// make all the initialize pop files on rtn, pass
extension > > > > > > > > > > > > > > > > > >
//

void PopFileInitialize (HWND hwnd, char *ext)
   {
   static char *szFilter[] =
            {
            "Text Files (*.PDF)", "*.pdf",
            "All Files (*.*)",    "*.*",
            ""
            } ;
   static char *szTFilter[] =
            {
            "Text Files (*.TR1)", "*.tr1",
            "All Files (*.*)",    "*.*",
            ""
            } ;

   ofn.lStructSize     = sizeof (OPENFILENAME) ;
   ofn.hwndOwner       = hwnd ;
   ofn.hInstance       = NULL ;
if (strcmp(ext,PDF) = = 0)
   {
   ofn.lpstrDefExt     = "pdf";
   ofn.lpstrFilter     = szFilter [0] ;
   }
else if (strcmp(ext,TR1) = = 0)
   {
   ofn.lpstrDefExt     = "tr1";
   ofn.lpstrFilter     = szTFilter [0] ;
   }
   ofn.nMaxCustFilter  = 0 ;
```

```
ofn.nFilterIndex    = 0 ;
ofn.lpstrFile       = NULL ;       // Set in Open and Close functions
ofn.nMaxFile        = _MAX_PATH ;
ofn.lpstrFileTitle  = NULL ;       // Set in Open and Close functions
ofn.nMaxFileTitle   = _MAX_FNAME + _MAX_EXT ;
ofn.lpstrInitialDir = NULL ;
ofn.lpstrTitle      = NULL ;
ofn.Flags           = 0 ;          // Set in Open and Close functions
ofn.nFileOffset     = 0 ;
ofn.nFileExtension  = 0 ;

ofn.lCustData       = 0L ;
ofn.lpfnHook        = NULL ;
ofn.lpTemplateName  = NULL ;
}


BOOL PopFileOpenDlg (HWND hwnd, LPSTR lpstrFileName, LPSTR lpstrTitleName)
{
BOOL rv; DWORD t;
ofn.hwndOwner       = hwnd ;
ofn.lpstrFile       = lpstrFileName ;
ofn.lpstrFileTitle  = lpstrTitleName ;
ofn.Flags           = OFN_CREATEPROMPT ;

rv = GetOpenFileName (&ofn) ;
t = CommDlgExtendedError();
return rv ;
}

BOOL PopFileSaveDlg (HWND hwnd, LPSTR lpstrFileName, LPSTR lpstrTitleName)
{
ofn.hwndOwner       = hwnd ;
ofn.lpstrFile       = lpstrFileName ;
ofn.lpstrFileTitle  = lpstrTitleName ;
ofn.Flags           = OFN_OVERWRITEPROMPT ;
```

```
    return GetSaveFileName (&ofn) ;
    }

//static long PopFileLength (WORD hFile)
//  {
//  long lCurrentPos = _llseek (hFile, 0L, 1) ;
//  long lFileLength = _llseek (hFile, 0L, 2) ;
//
//  _llseek (hFile, lCurrentPos, 0) ;
//
//  return lFileLength ;
//  }

protocol *PopFileRead (HWND hwndEdit, LPSTR lpstrFileName)
  {
  for (int i=0; i<ProtocolList->entries(); i++)
    {
    if (strcmp(strrchr(lpstrFileName, '\\')+1, ProtocolList->operator[](i).prot()->filename()) == 0)
      {
        hwndEdit = 0;
// Display Protocol in Windows
      return(ProtocolList->operator[](i).prot());
      }
    }
// Open any file not already in list...add to list and directory of pdf files if user requests it
  return(0);
  }
#ifdef notdef
BOOL PopFileWrite (HWND hwndEdit, LPSTR lpstrFileName)
  {
  HANDLE hBuffer ;
  WORD  hFile ;
  LPSTR  lpstrBuffer ;
  WORD  wLength ;

  if (-1 == (hFile = _lopen (lpstrFileName, OF_WRITE | OF_SHARE_EXCLUSIVE)))
    if (-1 == (hFile = _lcreat (lpstrFileName, 0)))
```

```
        return FALSE;

    wLength = GetWindowTextLength (hwndEdit);
    hBuffer = (HANDLE) SendMessage (hwndEdit, EM_GETHANDLE, 0, 0L);
    lpstrBuffer = (LPSTR) LocalLock (hBuffer);

    if (wLength != _lwrite (hFile, lpstrBuffer, wLength))
       {
       _lclose (hFile);
       return FALSE;
       }

    _lclose (hFile);
    LocalUnlock (hBuffer);

    return TRUE;
    }
#endif
→

////////////////////////////////////////////////////////////////////////
//   STAT.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"
//*************************************************************************
*****
// Generate Required data structure from user statistics selections
//
// Parameters:
//   field - pointer to protocol template field
//           (should only be called for fields with stats configured)
//
// Returns:
//        status - OK / NOTOK
//
```

```
//*************************************************************************
*****
stats *alloc_stat_structs(U32 type, field *f)
{
switch(type)
  {
  default:
  case NOSTATS:  return(0);        // No statistics to be kept for this protocol field
  case SUM:      return(new sum_stats(f->name()));
  case INC:      return(new inc_stats(f->name()));
  case SUMINC:   return(new suminc_stats(f->name()));
  case IDXSUM:   return(new idxsum_stats(f->name(), (32-f->shrbits())));
  case IDXINC:   return(new idxinc_stats(f->name(), (32-f->shrbits())));
  case IDXSUMINC: return(new idxsuminc_stats(f->name(), (32-f->shrbits())));
  case HASHSUM:  return(new hashsum_stats(f->name()));
  case HASHINC:  return(new hashinc_stats(f->name()));
  case HASHSUMINC:return(new hashsuminc_stats(f->name()));
  case IDXINCSUM: return(new idxincsum_stats(f->name(), (32-f->shrbits())));
  case HASHINCSUM:return(new hashincsum_stats(f->name()));
  case CSUMINC:  return(0);
  }
}
→

////////////////////////////////////////////////////////////////////////
//    XMIT.CPP
////////////////////////////////////////////////////////////////////////

#include "gen.hpp"

//
// IPX checksum incremental update routine (bitswapped)
//
extern U32 IpxCsumUpd(U32 value, U32 change);
#pragma aux IpxCsumUpd =      \
  "mov   ecx,FrameLen"     \
  "sub   ecx,ParseLen"     \
```

```
        "sub   ecx,ProtoParseLen" \
        "shl   ecx,+18H"          \
        "shr   ecx,+1cH"          \
        "rol   dx,cl"             \
        "add   ax,dx"             \
        "adc   ax,0"              \
        parm   [eax] [edx]        \
        modify [eax ecx edx]      \
        value  [eax]


U32 vary_ipxcsum::vary_value(U32 val) { return(IpxCsumUpd(val, operand)); }


-

////////////////////////////////////////////////////////////////////////
//    CHECKSUM.HPP
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Base Class definition for verifying and computing protocol checksums
////////////////////////////////////////////////////////////////////////
class CheckSum
{
public:
//
// Constructor
   CheckSum() {}
   virtual ~CheckSum() {}
//
// Member Functions
   virtual BOOL verify() { return(0); }   // Verify checksum
   virtual void compute() {}              // Compute checksum
};

////////////////////////////////////////////////////////////////////////
```

```
// Derived Class definition verifying and computing IP Checksums
//////////////////////////////////////////////////////////////////////////
// Verify IP Header Checksum in Network Byte Order...No byte swapping required
//    (Assumes Header Length is greater than 4 32-bit words)
//    (Returns 0 if checksum OK, non-zero otherwise)
//////////////////////////////////////////////////////////////////////////
extern U32 ChkIpCsum(S8 *h); // Clocks = 14 + #OptionBytes
#pragma aux ChkIpCsum =              \
    "mov   ebx,[edx]"          \
    "mov   eax,ebx"            \
    "and   ebx,+0fH"           \
    "mov   ecx,+4H[edx]"       \
    "sub   ebx,5"              \
    "jle   Header"             \
"Options:adc   eax,+10H[ebx*4+edx]"  \
    "dec   ebx"                \
    "jg    Options"            \
"Header: mov   ebx,+8H[edx]"       \
    "adc   eax,ecx"            \
    "mov   ecx,+0cH[edx]"      \
    "adc   eax,ebx"            \
    "mov   ebx,+10H[edx]"      \
    "adc   eax,ecx"            \
    "adc   eax,ebx"            \
    "adc   eax,0"              \
    "mov   ecx,eax"            \
    "shr   eax,16"             \
    "add   ax,cx"              \
    "adc   ax,0"               \
    "not   ax"                 \
    parm   [edx]               \
    modify [ebx ecx]           \
    value  [eax]

class CheckSum_IP: public CheckSum
{
public:
```

```
//
// Constructor
  CheckSum_IP() {}
//
// Verify IP Checksum (Return TRUE if OK, FALSE otherwise)
  BOOL verify() { return(ChkIpCsum(ParsePtr) == 0); }
//
// Compute IP checksum and insert into IP header
  void compute()
    {// (35/36 + 3*(Option DWORDS)) (20 --- 0.360 usecs; 60 --- 0.660 usecs)
    U16 *csum = ((U16 *)ParsePtr + 5); // Declare Pointer to Checksum Field
    *csum = 0;                  // Clear Checksum in IP Header
    *csum = (U16)ChkIpCsum(ParsePtr);  // Compute and Update Checksum
    }
private:
};

////////////////////////////////////////////////////////////////////////
// Derived Class definition verifying and computing IPX Checksums
////////////////////////////////////////////////////////////////////////
// Verify IPX Header Checksum in Network Byte Order...No byte swapping required
//    (Assumes Checksum Field is NOT 0xffff, and 30 > = Length Field < = 0xffff)
//    (Returns 0 if checksum OK, non-zero otherwise)
////////////////////////////////////////////////////////////////////////
extern U32 ChkIpxCsum(S8 *h); // Clocks = 5 + 7*(# Words)
#pragma aux ChkIpxCsum =          \
    "mov   ecx,[ebx]"          \
    "xor   eax,eax"            \
    "bswap ecx"                \
    "mov   edx,+1H"            \
    "inc   ecx"                \
    "shl   ecx,16"             \
    "shr   ecx,17"             \
"NxtWord:add   ax,[ebx+edx*2]"  \
    "adc   ax,0"               \
    "rol   ax,1"               \
    "inc   edx"                \
```

```
        "cmp   edx,ecx"          \
        "jl    NxtWord"          \
        parm   [ebx]             \
        modify [ecx edx]         \
        value  [eax]

class CheckSum_IPX: public CheckSum
{
public:
//
// Constructor
  CheckSum_IPX() {}
//
// Verify IPX Checksum (Return TRUE if OK, FALSE otherwise)
  BOOL verify() { return(*(U16 *)ParsePtr == 0xffff ||
                  *(U16 *)ParsePtr == ChkIpxCsum(ParsePtr)); }
//
// Compute IPX checksum and insert into IPX header
  void compute()
    {// (20/21 + 7*WORDS) (30 --- 1.26; 96 --- 3.57; 512 --- 18.13 usecs)
    *(U16 *)ParsePtr = (U16)ChkIpxCsum(ParsePtr);
    }
private:
};

extern "C" U32 csumBytes(S8 *Data, U32 Len);   // ASM TCP/IP Checksum Routine
extern "C" U32 csumPseudo(U32 Csum, S8 *IpHdr); // ASM PseudoHeader Checksum

//////////////////////////////////////////////////////////////////////
// Derived Class definition verifying and computing TCP Checksums
//////////////////////////////////////////////////////////////////////
class CheckSum_TCP: public CheckSum
{
public:
//
// Constructor
  CheckSum_TCP() {}
```

```
    //
    // Verify TCP Checksum (Return TRUE if OK, FALSE otherwise)
      BOOL verify()
        {
        U32 csum = csumBytes(ParsePtr, (FrameLen-ParseLen+ProtoParseLen)/8);
        return(csumPseudo(csum, ParseList.operator[](ParseLevel-2)) == 0);
        }
    //
    // Compute TCP checksum and insert into TCP header
      void compute()
        {
        ((U16 *)ParsePtr)[8] = 0;  // Clear checksum field
        U32 csum = csumBytes(ParsePtr, (FrameLen-ParseLen+ProtoParseLen)/8);
        ((U16 *)ParsePtr)[8] = (U16)csumPseudo(csum,
                                ParseList.operator[](ParseLevel-2));
        }
    private:
    };

    ////////////////////////////////////////////////////////////////////////
    // Derived Class definition verifying and computing UDP Checksums
    ////////////////////////////////////////////////////////////////////////
    class CheckSum_UDP: public CheckSum
    {
    public:
    //
    // Constructor
      CheckSum_UDP() {}
    //
    // Verify UDP Checksum (Return TRUE if OK, FALSE otherwise)
      BOOL verify()
        {
        U32 csum = csumBytes(ParsePtr, (FrameLen-ParseLen+ProtoParseLen)/8);
        return(csumPseudo(csum, ParseList.operator[](ParseLevel-2)) == 0);
        }
    //
    // Compute UDP checksum and insert into UDP header
```

```
    void compute()
      {
      ((U16 *)ParsePtr)[3] = 0;  // Clear Checksum field
      U32 csum = csumBytes(ParsePtr, (FrameLen-ParseLen+ProtoParseLen)/8);
      ((U16 *)ParsePtr)[3] = (U16)csumPseudo(csum,
                              ParseList.operator[](ParseLevel-2));
      }
};
-

////////////////////////////////////////////////////////////////////////
//    DECODE.HPP
////////////////////////////////////////////////////////////////////////

//
// Protocol Field Class Description
//
//
enum out_fmt   {BIN, DEC, HEX, HW, ASC };
enum rangetypes {       ODDNUMBERS=1, EVENNUMBERS, ALLNUMBERS};

class lookup;     // forward reference
class protocol;
class field
{
public:

// Constructors
  field(): fdwoff(0), fshl(0), fshr(0), ffmt(2),
        flflag(0), fname(0), fblen(8), reserved(0), fmult(1),
        fdspfield(1), fswap(1), fplen(0), ptr2stats(0), ptr2np(0),
        ptr2vary(0), ptr2csum(0), ptr2flt(0), ptr2rte(0) {}
  field(U32 o, U8 l, U8 r):
        ffmt(2), flflag(0), fname(0), reserved(0), fmult(1),
        fdspfield(1), fswap(1), fplen(0), ptr2stats(0), ptr2np(0),
        ptr2vary(0), ptr2csum(0), ptr2flt(0), ptr2rte(0)
    {
```

```
        fdwoff = o;
        fshl = l;
        fshr = r;
        fblen = 32 - fshr;
    }

// Destructor
  ~field()
    {
#ifdef notdef
    if (fname != 0)      delete []fname;
    if (ptr2stats != 0)  delete ptr2stats;
    if (ptr2np != 0)     delete ptr2np;
    if (ptr2vary != 0)   delete ptr2vary;
    if (ptr2csum != 0)   delete ptr2csum;
    if (ptr2flt != 0)    delete ptr2flt;
    if (ptr2rte != 0)    delete ptr2rte;
#endif
    }

// Overloaded operators
  field& operator=(const field& f)
    {
    fplen    = f.fplen;
    fblen    = f.fblen;
    fdwoff   = f.fdwoff;
    fshl     = f.fshl;
    fshr     = f.fshr;
    ffmt     = f.ffmt;
    flflag   = f.flflag;
    reserved = f.reserved;
    fmult    = f.fmult;
    fswap    = f.fswap;
    fdspfield= f.fdspfield;
    fname    = f.fname;
    ptr2stats= f.ptr2stats;
    ptr2np   = f.ptr2np;
```

```
        ptr2vary = f.ptr2vary;
        ptr2csum = f.ptr2csum;
        ptr2flt  = f.ptr2flt;
        ptr2rte  = f.ptr2rte;
        return(*this);
        }
    //
    // Routine to compute bit offset of field from start of protocol header
      U32 bitoffset() const     { return(fdwoff*8 + fshl); }
    //
    // Routine to read/write field offset value
      U32 offset() const        { return(fdwoff); }
      void offset(U32 o)        { fdwoff = o; }
    //
    // Routines to read/write left shift value
      U32 shlbits() const       { return(fshl); }
      void shlbits(U8 l)        { fshl = l; }
    //
    // Routines to read/write right shift value
      U32 shrbits() const       { return(fshr); }
      void shrbits(U8 r)        { fshr = r; }
    //
    // Routines to read/write field format value
      U8 format() const         { return(ffmt); }
      void format(U8 f)         { ffmt = f; }
    //
    // Routines to read/write display this field flag
      U8 dspfield() const       { return(fdspfield); }
      void dspfield(U8 f)       { fdspfield = f; }
    //
    // Routines to read/write field name
      S8 *name() const          { return(fname); }
      void name(S8 *n)
        {
        if (n != 0)
          {
          if (fname != 0) delete []fname;
```

```
            fname = new S8[strlen(n)+1];
            strcpy(fname, n);
            }
          }
//
// Routines to read/write header length flag
  U8 hdlen() const          { return(flflag); }
  void hdlen(U8 l)          { flflag = l; }
//
// Routines to read/write field length in bits
  U32 bitlen()              { return(fblen); }
  void bitlen(U32 l)        { fblen = l; }
//
// Routines to read/write field multiplier value (for display)
  U8 multiplier() const     { return(fmult); }
  void multiplier(U8 m)     { fmult = m; }
//
// Routines to read/write end of display line character
  U8 swap() const           { return(fswap); }
  void swap(U8 c)           { fswap = c; }
//
// Routines to read/write overall protocol length field
  U32 protlen() const       { return(fplen); }
  U32 protlen(U32 p)        { return(fplen = p); }
//
// Routines to read/write pointer to statistics collection object
  stats *statsptr() const   { return(ptr2stats); }
  void statsptr(stats *s)   { ptr2stats = s; }
//
// Routines to read/write pointer to lookup value validation object
  lookup *lookptr() const   { return(ptr2np); }
  void lookptr(lookup *l)   { ptr2np = l; }
//
// Routines to read/write pointer to vary field value object
  vary *varyptr() const     { return(ptr2vary); }
  void varyptr(vary *v)     { ptr2vary = v; }
//
```

```
// Routines to read/write pointer to checksum verification object
  CheckSum *csumptr() const  { return(ptr2csum); }
  void csumptr(CheckSum *c)  { ptr2csum = c; }
//
// Routines to read/write pointer to filter criteria object
  criteria *fltptr() const   { return(ptr2flt); }
  void fltptr(criteria *c)   { ptr2flt = c; }
//
// Routines to read/write pointer to Routing Table object
  RouteTable *rteptr() const  { return(ptr2rte); }
  void rteptr(RouteTable *r)  { ptr2rte = r; }
//
// Routine to extract field values using offset, left and right shift values
//    value = ((values at offset) << left shift) >> right shift
//
// Value could also be extracted by:
//    ((value & predetermined MASK) >> predetermined SHR)
// ... to eliminate a 4 clock left shift by cl on 486/Pentium
//
  U32 get_value()
    {
    U32 *ptr = (U32 *)(ParsePtr + fdwoff);
    U32 val = *ptr;
    if (fswap != 0)
      {
      val = wordswap(val);
      if (ptr2vary == 0)
        return((val << fshl) >> fshr);
      else   // This field needs its value varied
        return(*ptr = wordswap(ptr2vary->vary_value(val)));
      }
    else
      {
      if (ptr2vary == 0)
        return((val << fshl) >> fshr);
      else   // This field needs its value varied
        return(*ptr = ptr2vary->vary_value(val));
```

```
        }
      }
  //
  // Routine to check field value for validity
  //
    U32 value_ok(U32 value, U32 &i, protocol * &tp)
      {
      verify *v;
      if (ptr2np == 0)
        {
        ++i;
        return(TRUE);
        }

      if ((v = ptr2np->value_ok(value)) == 0)
        {
        ++i;
        return(FALSE);
        }
       else
        {
        if (tp == 0)
          tp = v->prot;
        i = v->nxtidx;
        return(TRUE);
        }
      }
  //
  // Routine to write field to designated output file
  //
    void out_to_file(FILE *fp)
      {
      fwrite(&fplen, sizeof(fplen), 1, fp);
      fwrite(&fblen, sizeof(fblen), 1, fp);
      fwrite(&fdwoff, (S8 *)&fname - (S8 *)&fdwoff, 1, fp);
  //
      U32 tmp;
```

```
        if (fname != 0)
          {
          tmp = strlen(fname) + 1;
          fwrite(&tmp, sizeof(tmp), 1, fp);  // Write out length of field name
          fwrite(fname, tmp, 1, fp);         // Write out field name
          }
        else
          {
          tmp = 0;
          fwrite(&tmp, sizeof(tmp), 1, fp);  // Write out length of field name
          }
    //
        tmp = (ptr2stats == 0) ? NOSTATS : ptr2stats->get_type();
        fwrite(&tmp, sizeof(tmp), 1, fp);  // Write out type of stats to gather
    //
        if (ptr2np == 0)
          {
          tmp = NOLOOKUP;
          fwrite(&tmp, sizeof(tmp), 1, fp);
          }
        else
          {
          tmp = ptr2np->get_type();
          fwrite(&tmp, sizeof(tmp), 1, fp);
          ptr2np->out_to_file(fp);       // Write out values/ranges in lookup
          }
        }
    //
    // Routine to set up existing field object from designated file
    //
      void get_from_file(FILE *fp)
        {
        U32 tmp;
        fread(&fplen, sizeof(fplen), 1, fp);
        fread(&fblen, sizeof(fblen), 1, fp);
        fread(&fdwoff, (S8 *)&fname - (S8 *)&fdwoff, 1, fp);
    //
```

```
        fread(&tmp, sizeof(tmp), 1, fp);   // Read length of field name
        if (tmp != 0)
          {
          fname = new S8[tmp];
          fread(fname, tmp, 1, fp);        // Read field name
          }
        else fname = 0;
        fread(&tmp, sizeof(tmp), 1, fp);   // Read type of stats to be gathered
        ptr2stats = alloc_stat_structs(tmp, this);
        fread(&tmp, sizeof(tmp), 1, fp);   // Read type of lookup structure
        if ((ptr2np = alloc_lookup_structs(tmp, this)) != 0)
          ptr2np->get_from_file(fp);       // Read all lookup values for field
        }

    //
    //
    //
      void OutlineField(HDC hdc, protocol *p, U32 dspbW) const;
      void OutlineFieldVal(HDC hdc, protocol *p, U32 dspbW, unsigned long val, unsigned char fmat,
    unsigned long bitwid) const;
      U32  field::FindF(U32 x, U32 y, U32 dspbW) const;

    // Data Representation
    private:
    // General purpose field descriptors
      U32       fplen;     // if not 0, value * field contents is FrameLen
      U32       fblen;     // field length in bits
      U32       fdwoff;    // field offset in bytes
      U8        fshl;      // Number of bits to shift left
      U8        fshr;      // Number of bits to shift right
      U8        ffmt;      // Field output display format
      U8        flflag;    // if TRUE this field contains length of option
      U8        reserved;  // not used...pad to align following fields
      U8        fmult;     // multiply value by this amount before display
      U8        fswap;     // Flag indicating the need to swap bytes and words
      U8        fdspfield; // display this field on output
      S8        *fname;    // user defined field name
```

```
    stats     *ptr2stats; // Pointer to derived statistics class
    lookup    *ptr2np;   // Pointer to derived lookup class (next_protocol)
    vary      *ptr2vary; // Pointer to Vary field value class
    CheckSum  *ptr2csum; // Pointer to Checksum class for current protocol
    criteria  *ptr2flt;  // Pointer to Filter Criteria class for this field
    RouteTable *ptr2rte;  // Pointer to Route Determination class
};

//
// Template class for defining fixed length protocol header fields
//
//
class protocol
{
public:
//
// Constructors
  protocol(): name_length(0), protocol_name(0), fname(0), num_fields(0),
        num_bits(0), out_flag(0), fs(0), dbW(32), opts(0), Rt(0) {}
//
  protocol(S8 *pn, S8 *fn, protocol *p=0): num_fields(0), num_bits(0),
    out_flag(0), fs(0), cur_field(0), dbW(32), Rt(0)
    {
    name_length=strlen(pn)+1;
    protocol_name = (S8 *)malloc(name_length);
    strcpy(protocol_name, pn);
    fname = (S8 *)malloc(strlen(fn)+1);
    strcpy(fname, fn);
    opts = p;  // Pointer to protocol for parsing options
    }
//
  protocol(int fd)
    {
    read(fd, &num_fields, sizeof(num_fields));
    fs = new field[num_fields];
    read(fd, fs, num_fields*sizeof(field));
    } // open file and init structure in memory
```

```
    // Destructor
      ~protocol()
        {
        if (protocol_name != 0)free(protocol_name);
        if (fname != 0)      free(fname);
        if (fs != 0)         delete []fs;
        if (opts != 0)       delete opts;
        if (Rt != 0)         delete Rt;
        }

    //
    // Routines to read/write protocol name length field
      U32 pnamelen() const       { return(name_length); }
      void pnamelen(U32 nl)      { name_length = nl; }
    //
    // Routines to read/write protocol name field
      S8 *pname() const          { return(protocol_name); }
      void pname(S8 *n)
        {
        if (n != 0)
          {
          if (protocol_name != 0) delete []protocol_name;
          protocol_name = new S8[strlen(n)+1];
          strcpy(protocol_name, n);
          }
        }
    //
    // Routines to read/write protocol file name
      S8 *filename() const       { return(fname); }
      void filename(S8 *n)
        {
        if (n != 0)
          {
          if (fname != 0) delete []fname;
          fname = new S8[strlen(n)+1];
          strcpy(fname, n);
          }
```

```
    }
  //
  // Routines to read/write number of fields
    U32 numfields() const      { return(num_fields); }
    void numfields(U16 nf)     { num_fields = nf; }
  //
  // Routines to read/write current field
    U32 curfield() const       { return(cur_field); }
    void curfield(U32 idx)     { cur_field = (U16)idx; }
  //
  // Routines to read/write number of bits
    U32 numbits() const        { return(num_bits); }
    void numbits(U32 nb)       { num_bits = nb; }
  //
  // Routines to read/write fields pointer
    field *fieldptr() const    { return(fs); }
    field *fieldptr(U32 i) const{ return(&fs[i]); } // Indexing
    void fieldptr(field *f)    { fs = f; }
  //
  // Routines to read/write options pointer
    protocol *options() const  { return(opts); }
    void options(protocol *o)  { opts = o; }
  //
  // Routines to read/write pointer to Routing Table
    RouteTable *RouteTbl() const{ return(Rt); }
    void RouteTbl(RouteTable *r){ Rt = r; }
  //
  // Routine to clear the protocol already output flag
    void clear_out_flag()      { out_flag = 0; }
  //
  // Routines to read/write the display bit width field
    U32 dspbW() const          { return(dbW); }
    void dspbW(U32 bW)         { dbW = bW; }
  //
  // Routine to set current field to previous
  //
    U32 prevfield()
```

```
        {
        if (cur_field > 0)
          --cur_field;
        else
          return(FALSE);
        return(TRUE);
        }
    //
    // Routine to set current field to next
    //
      U32 nextfield()
        {
        if (cur_field < (num_fields-1))
          ++cur_field;
        else
          return(FALSE);
        return(TRUE);
        }
    //
    // Routine to output entire protocol to file...Save As passes parameter fp
    //
      void out_to_file(FILE *fp=0)
        {
        if (out_flag != 0) return;
        out_flag = 1;
        if (fp != 0 || (fp = fopen(fname, "wb")) != 0)
          {
          fwrite(&name_length, sizeof(name_length), 1, fp);
          fwrite(protocol_name, name_length, 1, fp);
          fwrite(&num_bits, sizeof(num_bits), 1, fp);
          fwrite(&num_fields, sizeof(num_fields), 1, fp);
          fwrite(&num_fields, sizeof(num_fields), 1, fp);
          fwrite(&dbW, sizeof(dbW), 1, fp);
    //
    //      Write out each field
    //
          for (U32 i=0; i<num_fields; i++)
```

```
            fs[i].out_to_file(fp);
        //
        //      Write out option name length and any options
        //
        U32 tmp = (opts == 0) ? 0 : opts->name_length;
        fwrite(&tmp, sizeof(tmp), 1, fp);
        if (opts != 0)
          {
          fwrite(opts->protocol_name, opts->name_length, 1, fp);
          fclose(fp);
          opts->out_to_file();
          }
        else
          fclose(fp);
        }
      }
    //
    // Routine to initialize a protocol from a file
    //
     void get_from_file(FILE *fp);
    //
    // Routines for configuration and display purposes
    //
     void setupOutlineCoords();
     void setupOutlineCoordsMult(int continu);
     void OutlineProtocol(HDC hdc) const;
     void OutlineProtocolVal(HDC hdc) const;
     void FindField(HWND hwnd, U32 x, U32 y);
     void add_field(U32 idx, protocol *p);
     void add_field_after(U32 idx, protocol *ptl);
     void delete_field(U32 idx,protocol *ptl);
    //
    // Data fields
    private:
     U32      name_length;   // Length of protocol name in bytes
     S8       *protocol_name; // Protocol name in Ascii
     S8       *fname;        // File name in Ascii
```

```
    U32       num_bits;      // # of bits in fixed portion of protocol header
    U16       num_fields;    // # of fields used to describe protocol header
    U16       cur_field;     // Index of last field displayed
    U32       out_flag;      // Protocol has been output 'flag'
    U32       dbW;           // Display bit width (i.e. #bits per line)
    field     *fs;           // Fields for fixed portion of protocol header
    protocol  *opts;         // Pointer to protocol for parsing options
    RouteTable *Rt;          // Pointer to Protocol Route Table
};

extern void create_templates();
extern protocol *ParseProtocol(protocol *p);
extern protocol *ParseProtocol2(protocol *p);

extern protocol *DataPtlPtr, *PadPtlPtr;
→
→

//////////////////////////////////////////////////////////////////////
//   FILTER.HPP
//////////////////////////////////////////////////////////////////////

enum FilterStuff { FILTER_FRAME, PASS_FRAME, MAX_FILTERS=20 };
extern U32 FrameFilterStatus;
#define MODIFY  0
#define INSERT  1

class Filters;
class channel;
//////////////////////////////////////////////////////////////////////
// Filter Channel Criteria Class
//   Consists of:
//      Index     - Index # of this filter channel criteria
//      ChPtr     - Pointer back to filter channel of this criteria
//      Ranges    - Pointer to lookup class containing criteria values
//      Ptl       - Pointer to associated protocol class
//      Fld       - Pointer to associated protocol field class
```

```
////////////////////////////////////////////////////////////////////
class criteria              // Filter Channel Criteria Class
{
public:
//
// Constructor
  criteria(): Index(0), ChPtr(0), Ranges(0), Ptl(0), Fld(0) { }
//
// Destructor
  ~criteria()  {  if (Ranges != 0) delete Ranges;  }
//
// Member Functions
//
// Allows pre-declared criteria class to be filled in
  void FillInCriteria(U32 i, channel *c, lookup *r, protocol *p, field *f) {
    Index = i; ChPtr = c; Ranges = r; Ptl = p; Fld = f; }
//
// Applies the configured filter channel criteria to a received field value
  void ApplyFilterCriteria(U32 value);
//
private:
  U32      Index;  // Zero-based index number for this filter criteria
  channel  *ChPtr; // Pointer to Associated Filter Class
  lookup   *Ranges; // Pointer to lookup instance for this filter criteria
  protocol *Ptl;   // Required for Configuration/Display purposes only
  field    *Fld;   // Required for Configuration/Display purposes only
};

////////////////////////////////////////////////////////////////////
// Filter Channel Class
//   NextCriteriaIndex - Index of next filter criteria to apply
//   TotalCriteria     - Number of criteria that define this filter
//   Criteria          - Pointer to defining Criteria classes
//   ChannelName       - Pointer to Channel Name string
//   FramesAccepted    - # of Frames that passed filter channel criteria
//   FrameBitsAccepted - # of Frame Bits that passed filter channel criteria
////////////////////////////////////////////////////////////////////
```

```
class channel
{
public:
// Constructors/Destructor
  channel(U32 nci=0, U32 tot=0, criteria *ptr=0, S8 *name=0)
    : NextCriteriaIndex(nci), TotalCriteria(tot), Criteria(ptr),
      ChannelName(name), FramesAccepted(0), FrameBitsAccepted(0)  { }
  ~channel();       // must be defined in .cpp file to compile
//
// Equivalence operator ... required by WCPtrOrderedVector<channel> class
    int operator==(const channel& f) const
      { return(strcmp(ChannelName, f.ChannelName) == 0); }
//
// Member Functions
//
// Disables channel criteria application by indicating all criteria applied
  void Disable() { NextCriteriaIndex = TotalCriteria; }
//
// Load/Store Functions for NextCriteriaIndex field
  U32 NciValue() { return(NextCriteriaIndex); }
  void NciValue(U32 value) { NextCriteriaIndex = value; }
//
// Uses verify class pointer to update NextCriteriaIndex
// if the channel criteria status (prot) is not PASS_FRAME
//   returns FALSE
// else
//   Returns TRUE and updates the # of frames/bits accepted thru this channel
//
  void Update(verify *v);
//
private:
  U32     NextCriteriaIndex; // Index of next channel criteria to be applied
  U32     TotalCriteria;   // For display and disabling filter evaluation
  criteria *Criteria;      // Pointer to array of channel criteria
  S8     *ChannelName;     // Pointer to name of channel
  U64     FramesAccepted;  // Number of Frames Accepted by this channel
  U64     FrameBitsAccepted; // Number of Bits Accepted by this channel
```

```
};

////////////////////////////////////////////////////////////////////
//    Active Filters Class (e.g. CfgFilters, DisplayFilters, ...)
////////////////////////////////////////////////////////////////////
class Filters
{
public:
//
// Constructor
  Filters()
    {
    FilterStat = PASS_FRAME;
    Filters = new WCPtrOrderedVector<channel>;
    }
//
// Destructor
  ~Filters() { if (Filters != 0) delete Filters; }
//
// Member Functions
//
// Number of Configured Filters
  U32 entries() const { return(Filters->entries()); }
//
// Insert filter into FilterTable
  int insert(channel *f)
    {
    FilterStat = FILTER_FRAME;
    return(Filters->insert(f));
    }
//
// Remove filter from FilterTable
  channel *remove(channel *chn)
    {
    if ((chn = Filters->remove(chn)) != 0 && entries() == 0)
      FilterStat = PASS_FRAME;
    return(chn);
```

```
    }
//
  U32 FrameFilterStatus() { return(FilterStat); }
  void FrameFilterStatus(U32 Status) { FilterStat = Status; }
//
// Reset all Configured Filters
  void reset()
    {
    FilterStat = entries() ? FILTER_FRAME : PASS_FRAME;
    for (U32 j=0; j<entries(); j++)
      Filters->operator[](j)->NciValue(0);
    }
private:
  U32 FilterStat;
  WCPtrOrderedVector<channel> *Filters; // Pointer to Configured Filters Table
};

extern Filters CfgFilters; // Configured Filters Object

////////////////////////////////////////////////////////////////////////
//   GEN.HPP
////////////////////////////////////////////////////////////////////////

#define INCLUDE_COMMDLG_H     1
#include <windows.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.hpp>
#include <conio.h>
#include <time.h>
#include <iomanip.h>
#include <wchash.h>
#include <wcvector.h>
#include <io.h>
#include <dos.h>
```

```
#include "pa.hpp"
//
// Menu Item identifiers for now
//
//
#define MAIN_MENU_POS    1
#define HELLO_MENU_POS   2
#define RECT_MENU_POS    1

#define IDM_NEWHELLO     10
#define IDM_NEWRECT      11
//#define IDM_CLOSE       12
//#define IDM_EXIT        13

#define IDM_BLACK        20
#define IDM_RED          21
#define IDM_GREEN        22
#define IDM_BLUE         23
#define IDM_WHITE        24

#define IDM_TILE         30
#define IDM_CASCADE      31
#define IDM_ARRANGE      32
#define IDM_CLOSEALL     33


#define IDM_NEW          41
#define IDM_OPEN         42
#define IDM_SHOW         43
#define IDM_SAVE         44
#define IDM_SAVEAS       45
#define IDM_CLOSE        46
#define IDM_EXIT         47
#define IDM_ITEM1        50
#define IDM_ITEM2        60

#define IDM_FIRSTCHILD  100
```

```
// TEMP Menu bar stuff for demo
#define IDM_MACPTL     101
#define IDM_FRAMENAME  102
#define IDM_FRAMEFILE  103
#define IDM_NEXTFRAME  104
#define IDM_PREVFRAME  105
#define IDM_RUN        106
#define IDM_ALYZERCLOSE 107
#define IDM_TRACEFILEOPEN 110
#define IDM_TRACEFILENEW 111

/*
#define IDM_NEW        1
#define IDM_OPEN       2
#define IDM_SHOW       3
#define IDM_SAVE       4
#define IDM_SAVEAS     5
#define IDM_CLOSE      6
#define IDM_EXIT       7
*/

//
// General Purpose typedefs for various data types
//
//
typedef unsigned char       U8;
typedef char                S8;
typedef unsigned short      U16;
typedef signed short        S16;
typedef unsigned long       U32;
typedef signed long         S32;

enum status  { NOTOK, OK };


extern U32 ParseLevel;
extern WCValVector<S8 *> ParseList;
```

```
extern S8 *FramePtr, *ParsePtr;
extern U32 HwLen, FrameLen, ParseLen, ProtoParseLen, HeaderLen;
extern U32 SrcIntf, IntfTypes[256];
extern S8 bs[256];
//
// Data and code for efficiently performing byte swapping
//
//
// Generic 2 byte swap routine (inline assembly)
//
//
extern U16 byteswap(U16 w);
#pragma aux byteswap = \
    "rol  ax,8" \
    parm  [ax] \
    modify [ax] \
    value  [ax]
//
// Generic 2 word swap routine (inline assembly)
//
//
extern U32 wordswap(U32 d);
#pragma aux wordswap = \
    "bswap eax" \
    parm  [eax] \
    modify [eax] \
    value  [eax]
//
// Function templates for implementing min() and max() functions
//
//
template < class TYPE >
inline TYPE min(TYPE a, TYPE b)
{
return((a < b) ? a : b);
}
//
```

```
template < class TYPE >
inline TYPE max(TYPE a, TYPE b)
{
return((a > b) ? a : b);
}
//

template < class TYPE >
inline TYPE *swapminmax(TYPE *a, TYPE *b)
{
if (a.minval < a.maxval) return(a);
b.minval = a.maxval;
b.maxval = a.minval;
return(b);
}

//
// 64 bit numbers class and associated pragmas
//
class U64;
extern void add64(U64 *u, U32 val);
#pragma aux add64 =                \
     "add   [edx],esi"             \
     "adc   +4H[edx],dword ptr 0"  \
     parm   [edx] [esi]

extern void sub64(U64 *u, U32 val);
#pragma aux sub64 =                \
     "sub   [edx],esi"             \
     "sbb   +4H[edx],dword ptr 0"  \
     parm   [edx] [esi]

class U64
{
// constructors
public:
   U64(): hi(0), lo(0) {}
```

```
        U64(U32 l): hi(0) { lo = l; }
        U64(U32 h, U32 l) { hi = h; lo = l; }

        // destructor
        ~U64() {}

        // Member functions
        //  void print() const;  // upto 99,999,999,999 (100 billion) (14 characters)
        U32 loval()  { return(lo); }
        U32 hival()  { return(hi); }
        U32 sum()   { U32 h = hi; hi = 0; lo += h; if (hi != 0) ++lo; return(lo); }
        void clear()        { lo = hi = 0; }

        U64 *init(U32 value) { lo = value; return(this); }

        // overloaded operators
        U64 operator++()     { add64(this, 1); return(*this);          }
        U64 operator--()     { sub64(this, 1); return(*this);          }
        U64 operator+=(U32 sz) { add64(this, sz); return(*this);       }
        U64 operator-=(U32 sz) { sub64(this, sz); return(*this);       }
        U64& operator=(U32 sz) { hi = 0; lo = sz; return(*this);       }
        U64 operator+(U32 sz)  { add64(this, sz); return(*this);       }
        U64 operator-(U32 sz)  { sub64(this, sz); return(*this);       }
        int operator==(const U64 &u) const { return(lo == u.lo && hi == u.hi); }


    // data representation
    private:
      U32 lo, hi;
    };

    #include "stat.hpp"
    #include "lookup.hpp"
    #include "xmit.hpp"
    #include "checksum.hpp"
    #include "filter.hpp"
    #include "route.hpp"
```

```
#include "decode.hpp"
#include "pcols.hpp"

////////////////////////////////////////////////////////////////////////
//    LOOKUP.HPP
////////////////////////////////////////////////////////////////////////

//*************************************************************************
****
//*************************************************************************
****
// Lookup Next Protocol Base Class Definition
//    Virtual member functions
//*************************************************************************
****
//*************************************************************************
****
class protocol;

class verify
{
public:
//
// Constructors
   verify(): minval(0), maxval(0), okbits(0), nxtidx(-1), xlat(0) {}
//
   verify(protocol *p, S32 indx, U32 minv, U32 maxv, U32 bits, S8* xl)
      {
      prot = p;
      minval = minv;
      maxval = maxv;
      okbits = bits;
      nxtidx = indx;
      xlat   = xl;
      }
//
   verify(const verify& t)
```

```
        {
        prot   = t.prot;
        minval = t.minval;
        maxval = t.maxval;
        okbits = t.okbits;
        nxtidx = t.nxtidx;
        xlat   = t.xlat;
        }
    //
    // destructor
      ~verify() {}
    //
    // overloaded operators
      int operator==(const verify &t) const
                { return(minval >= t.minval && maxval <= t.maxval); }
    //
      int operator <(const verify &t) const   { return(maxval < t.minval); }
    //
      verify & operator =(const verify &t)
        {
        prot   = t.prot;
        nxtidx = t.nxtidx;
        minval = t.minval;
        maxval = t.maxval;
        okbits = t.okbits;
        xlat   = t.xlat;
        return(*this);
        }
    //
    // Routine to output verify object to file
      void out_to_file(FILE *fp) const;
    //
    // Routine to setup verify object from file
      void get_from_file(FILE *fp);
    //
    // data representation (PUBLIC DATA)
      protocol *prot;
```

```
    S32     nxtidx;
    U32     minval;
    U32     maxval;
    U32     okbits;
    S8     *xlat;
};

enum lookuptypes{ NOLOOKUP, ONEVALUE, ARRAY, HASH, TREE };
////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////
class protocol;
class lookup
{
public:
// No constructors !!!
//
// Destructor
   virtual ~lookup() {}                          // default behavior
//
// Routine to insert verify structure into lookup class
    virtual BOOL insert_protocol(protocol *p, S32 idx, U32 minv=0, U32 maxv=0, U32 bits=0, S8
*xlat=0)
        {
        p = 0;
        idx = 0;
        minv = maxv = bits = 0;
        xlat = 0;
        return(TRUE);
        }
      virtual U32 entries() { return(0); }    // default behavior
      virtual verify *find_index(U32 idx) { idx=0; return(0); }
      virtual verify *inc_index(U32 insertidx) {insertidx=0; return(0); }
      virtual verify *dec_index(U32 deleteidx) {deleteidx=0; return(0); }
        virtual BOOL insert_entry(protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32
okbits=0, S8 *xlat=0)
        {
```

```
        prot = 0;
        nxtidx = 0;
        minval = maxval = okbits = 0;
        xlat = 0;
        return(TRUE);
        }
      virtual BOOL delete_entry(U32 idx, U32 min=0)
        {
        idx = min = 0;
        return(TRUE);
        }
      virtual BOOL modify_entry(U32 idx, protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32
    okbits=0, S8 *xlat=0)
        {
        prot = 0;
        idx = 0;
        nxtidx = 0;
        minval = maxval = okbits = 0;
        xlat = 0;
        return(TRUE);
        }
      virtual protocol *next_protocol(U32 value) { value = 0; return(0); }  // default behavior
      virtual verify *value_ok(U32 value) { value = 0; return(0); }
      virtual U32 get_type() { return(NOLOOKUP); }
      virtual void out_to_file(FILE *fp) const { fp = 0; }
      virtual void get_from_file(FILE *fp) { fp = 0; }
    protected:

    private:
    };

////////////////////////////////////////////////////////////////////////
// Derived Single Entry (Next Protocol) Lookup Class Definition
//    For use where the current protocol has only 1 upper level protocol
//
// Virtual member functions
// insert_protocol(protocol *p)     - Insert protocol pointer
```

```
// protocol *next_protocol(U32 value) - returns protocol pointer
//////////////////////////////////////////////////////////////////////
class lookup_valu: public lookup
{
public:
//
// Constructor
  lookup_valu()
    {
    range = new verify;
    range->prot   = 0;
    range->nxtidx = -1;
    entrees = 0;
    }
//
// Destructor
  ~lookup_valu()  { delete range; entrees=0; }

//
// return number of entries in derived class
  U32 entries()   { return(entrees); }
//
//
  verify *find_index(U32 idx)     { idx = idx; return(range); }
//
//
  verify *inc_index(U32 idx)
    { if (idx <= range->nxtidx) range->nxtidx++; return(0); }
//
//
  verify *dec_index(U32 idx)
    {if (idx <=range->nxtidx&&range->nxtidx!=0) --range->nxtidx; return(0);}
//
// Insert field range structure
//
  BOOL insert_entry(protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32 okbits=0, S8
*xlat=0)
```

```
        {
        if (minval > maxval)
          {
          U32 tmp = minval;
          minval = maxval;
          maxval = tmp;
          xlat = 0;
          }
        range->prot   = prot;
        range->minval = minval;
        range->maxval = maxval;
        range->okbits = okbits;
        range->nxtidx = nxtidx;
        entrees=1;
        return(TRUE);
        }
    //
    // Delete field range structure
    //
    BOOL delete_entry(U32 idx, U32 min=0)
        {
        if (idx >= entries())          // Only delete existing entries
            return(FALSE);
        min=entrees=0;
        return(TRUE); //tptr->removeAt(idx));
        }
    //
    // Modify (update) existing field range structure values
    //
    BOOL modify_entry(U32 idx, protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32
okbits=0, S8 *xlat=0)
        {
        xlat = 0;
        if (idx >= entries())
          return(FALSE);
        if (minval > maxval)
          {
```

```
        U32 tmp = minval;
        minval = maxval;
        maxval = tmp;
        }
    range->prot   = prot;
    range->minval = minval;
    range->maxval = maxval;
    range->okbits = okbits;
    range->nxtidx = nxtidx;
    return(TRUE);
    }
//
// Inserts pointer to protocol into prot
  BOOL insert_protocol(protocol *prot, S32 idx, U32 minv=0, U32 maxv=0, U32 bits=0)
    {
    range->prot   = prot;
    range->nxtidx = idx;
    range->minval = minv;
    range->maxval = maxv;
    range->okbits = bits;
    return(TRUE);
    }
//
// Returns pointer to protocol class
  protocol *next_protocol(U32 value=0) { value = value; return(range->prot); }
//
// Verify that value is within specified range
   verify *value_ok(U32 value)
    {
    if (value >= range->minval && value <= range->maxval)
      {
      if (value & 1) // It's an odd number
        {
        if (range->okbits & 1)        // ODD and ALL numbers ok
          return(range);
        }
      else        // It's an even number
```

```
            {
            if (range->okbits > 1)   // EVEN and ALL numbers ok
              return(range);
            }
          }
        return(0);
        }
    //
    // return lookup type
      U32 get_type() { return(ONEVALUE); }
    //
    // Output lookup object to file
      void out_to_file(FILE *fp) const { if (entrees == 1) range->out_to_file(fp); }
    //
    // setup lookup structure from file
      void get_from_file(FILE *fp) { entrees = 1; range->get_from_file(fp); }
    //
    // Data Representation
    private:
      verify *range;    // Pointer to protocol class
      U32    entrees;   // Indicates whether the entry has been setup
    };

    ////////////////////////////////////////////////////////////////////////
    // Derived Multiple Entry (Next Protocol) Table Lookup Class Definition
    //    For use where the values are small in size and number and closely
    //    spaced. The IP protocol field is an excellent example of this
    //    case. The numbers range from 0 to 255 so an array of pointers
    //    requires 256 * sizeof(protocol *) or 1024 bytes.
    //
    //    Virtual member functions
    //      insert_protocol(protocol *p)      - Insert protocol into lookup
    //      protocol *next_protocol(U32 value) - Returns pointer to next protocol
    ////////////////////////////////////////////////////////////////////////
    class lookup_vect: public lookup
    {
    public:
```

```
//
// Constructor
  lookup_vect(U32 maxsz=256)
    {
    size  = maxsz;
    array = new verify[size+1];
    memset(array, 0, (size+1)*sizeof(verify));
    }
//
// Destructor
  ~lookup_vect()
    {
    memset(array, 0, (size+1)*sizeof(verify));
    delete []array;
    entrees=0;
    }
//
// Member functions
  U32 entries()
    {
    for (U32 i=0, num=0; i<size; i++)
      if (array[i].okbits != 0)
        ++num;
    return(num);
    }
//
//
  verify *find_index(U32 idx)
    {
    for (U32 i=0, j=0, num=entries(); i<size&&j<num; i++)
      if (array[i].okbits != 0 && idx == j++)
        return(&array[i]);
    return(0);
    }
//
//
  verify *inc_index(U32 insertidx)
```

```
        {
        for (U32 i=0; i<size; i++)
          if ((array[i].okbits != 0 ) &&
            (insertidx <= array[i].nxtidx))
            array[i].nxtidx++;
        return(0);
        }
    //
    //
    verify *dec_index(U32 deleteidx)
        {
        for (U32 i=0; i<size; i++)
          if ((array[i].okbits != 0 ) &&
            (deleteidx <= array[i].nxtidx))
            array[i].nxtidx--;
        return(0);
        }

    //
    // Insert field range structure into binary tree
    //
    BOOL insert_entry(protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32 okbits=0, S8
    *xlat=0)
        {
        xlat = 0;
        if (minval >= size || array[minval].okbits != 0)
          return(FALSE);
        if (minval > maxval)
          {
          U32 tmp = minval;
          minval = maxval;
          maxval = tmp;
          }
        for (U32 i = minval; i <= maxval; i++)
          {
          if (array[i].okbits != 0)
            {
```

```
            if (
               ((array[i].minval > = minval) && (array[i].minval < = maxval)) | |
               ((array[i].maxval > = minval) && (array[i].maxval < = maxval)) )
               return(FALSE);
            }
          }
        array[minval].prot   = prot;
        array[minval].minval = minval;
        array[minval].maxval = maxval;
        array[minval].okbits = okbits;
        array[minval].nxtidx = nxtidx;
        + + entrees;
        return(TRUE);
        }
    //
    // Delete field range structure from binary tree
    //
      BOOL delete_entry(U32 idx, U32 min)
        {
        if (min > = size)          // Only delete existing entries
          return(FALSE);
        array[min].okbits = 0;
        --entrees;
        idx=idx;
        return(TRUE);
        }
    //
    // Modify (update) existing fieled range structure values
    //
      BOOL modify_entry(U32 idx, protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32
    okbits=0, S8 *xlat=0)
        {
        xlat = 0;
        if (idx > = size)
          return(FALSE);
        if (minval > maxval)
          {
```

```
            U32 tmp = minval;
            minval = maxval;
            maxval = tmp;
            }
        array[minval].prot   = prot;
        array[minval].minval = minval;
        array[minval].maxval = maxval;
        array[minval].okbits = okbits;
        array[minval].nxtidx = nxtidx;
        return(TRUE);
        }

    //
    // Insert protocol pointer into lookup table at element value
      BOOL insert_protocol(protocol *p, S32 idx, U32 minv=0, U32 maxv=0, U32 bits=0)
        {
        maxv = min(maxv, size);
        for (U32 i=minv; i<=maxv; i++)
          {
          if (((i & 1) && (bits & 1)) || ((i & 1) == 0 && bits > 1))
            {
            array[i].prot = p;
            array[i].nxtidx = idx;
            array[i].minval = minv;
            array[i].maxval = maxv;
            array[i].okbits = bits;
            }
          }
        return(FALSE);
        }
    //
    // Return protocol pointer at lookup table element value
      protocol *next_protocol(U32 value) { return(array[min(value, size)].prot); }
    //
    //
      verify *value_ok(U32 value)
        {
```

```
        if (value > size)
          return(0);
        else
          {
          verify *v = &array[value];
          if (v->okbits != 0)
            {
            if (value >= v->minval && value <= v->maxval)
              {
              if (value & 1) // It's an odd number
                {
                if (v->okbits & 1)          // ODD and ALL numbers ok
                  return(v);
                }
              else          // It's an even number
                {
                if (v->okbits > 1)   // EVEN and ALL numbers ok
                  return(v);
                }
              }
            return(0);     // Value was not found
            }
          }
//
// return lookup type
  U32 get_type()     { return(ARRAY); }
//
// Output lookup to file
    void out_to_file(FILE *fp) const
      {
      for (U32 i=0, num=0; i<size; i++)
        if (array[i].okbits != 0)
          ++num;
      fwrite(&num, sizeof(num), 1, fp);      // Write out number of valid array entries
      for (i=0; i<size; i++)
        if (array[i].okbits != 0)
```

```
            array[i].out_to_file(fp);        // Write out single range
        }
    //
    // Setup lookup from file
    void get_from_file(FILE *fp)
        {
        U32 num;
        fread(&num, sizeof(num), 1, fp);     // Write out number of valid array entries
        for (U32 j=0; j<num; j++)
          {
          verify v;
          v.get_from_file(fp);
          v.maxval = min(v.maxval, size);
          for (U32 i=v.minval; i<=v.maxval; i++)
            {
            if (((i & 1) && (v.okbits & 1)) || ((i & 1) == 0 && v.okbits > 1))
              {
              array[i].prot = v.prot;
              array[i].nxtidx = v.nxtidx;
              array[i].minval = i;
              array[i].maxval = i;
              array[i].okbits = v.okbits;
              }
            }
          }
        }
    //
    // Data Representation
    private:
      U32    size;   // Array size in entries
      U32    entrees; // Number of entries in array
      verify *array;  // Array of pointers to protocol classes
    };


    ////////////////////////////////////////////////////////////////////
    // Derived binary tree class for storing field ranges, indices and protocols
```

```
//
////////////////////////////////////////////////////////////////////
class lookup_tree: public lookup
{
public:
//
// Constructor
  lookup_tree(U32 treesz=WCDEFAULT_VECTOR_LENGTH)
            { tptr = new WCValSortedVector<verify>(treesz); }
//
// Destructor
  ~lookup_tree()    { delete tptr; }
//
// Return number of field ranges stored in binary tree
  U32 entries()     { return(tptr->entries()); }
//
// Return pointer to field range structure at specified index
  verify *find_index(U32 idx)
    { return((idx < entries()) ? &tptr->operator[](idx) : 0); }
//
//
  verify *inc_index(U32 insertidx)
    {
    for (U32 i=0; i<entries(); i++)
      {
      verify *t = find_index(i);
      if (insertidx <= t->nxtidx)
        t->nxtidx++;
      }
    return(0);
    }
//
//
  verify *dec_index(U32 deleteidx)
    {
    for (U32 i=0; i<entries(); i++)
      {
```

```
            verify *t = find_index(i);
            if (deleteidx <= t->nxtidx)
              t->nxtidx--;
            }
        return(0);
        }

    //
    // Insert field range structure into binary tree
      BOOL insert_entry(protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32 okbits=0, S8
*xlat=0)
        {
    //    swapminmax(minval, maxval);
        if (minval > maxval)
          {
          U32 tmp = minval;
          minval = maxval;
          maxval = tmp;
          }
        for (U32 i=0; i<entries(); i++)
          {
          verify *t = find_index(i);
          if (
            ((t->minval >= minval) && (t->minval <=maxval)) ||
             ((t->maxval >= minval) && (t->maxval <=maxval)) )
            return(FALSE);
          }
        verify r(prot, nxtidx, minval, maxval, okbits, xlat);
        tptr->insert(r);
        return(TRUE);
        }
    //
    // Delete field range structure from binary tree
      BOOL delete_entry(U32 idx, U32 min=0)
        {
        if (idx >= entries())          // Only delete existing entries
          return(FALSE);
```

```
        min = min;
        return(tptr->removeAt(idx)!=0);
        }
    //
    // Modify (update) existing fieled range structure values
      BOOL modify_entry(U32 idx, protocol *prot, S32 nxtidx, U32 minval=0, U32 maxval=0, U32
    okbits=0, S8 *xlat=0)
        {
        if (idx >= entries())
          return(FALSE);
        if (minval > maxval)
          {
          U32 tmp = minval;
          minval = maxval;
          maxval = tmp;
          }
        verify *vptr = &tptr->operator[](idx);
        vptr->prot   = prot;
        vptr->minval = minval;
        vptr->maxval = maxval;
        vptr->okbits = okbits;
        vptr->nxtidx = nxtidx;
        vptr->xlat   = xlat;
        return(TRUE);
        }
    //
    // Return protocol pointer in binary tree with specified value
      protocol *next_protocol(U32 value, S32& idx)
        {
        verify srch(0, 0, value, value, 0, 0), t;
        idx = -1;
        if (tptr->find(srch, t) == 0)
          return(0);
        idx = t.nxtidx;
        return(t.prot);
        }
      verify *value_ok(U32 value)
```

```
        {
        static verify t;
        verify srch(0, 0, value, value, 0, 0);
        if (tptr->find(srch, t) != 0)
          {
          if (value & 1) // It's an odd number
            {
            if (t.okbits & 1)          // ODD and ALL numbers ok
              return(&t);
            }
          else          // It's an even number
            {
            if (t.okbits > 1)   // EVEN and ALL numbers ok
              return(&t);
            }
          }
        return(0);
        }
    //
    // return lookup type
      U32 get_type()     { return(TREE); }
    //
    // output lookup to file
      void out_to_file(FILE *fp) const
        {
        U32 num = tptr->entries();
        fwrite(&num, sizeof(num), 1, fp);      // Write out number of sorted list entries
        for (U32 i=0; i<num; i++)
          tptr->operator[](i).out_to_file(fp);       // Write out single range
        }
    //
    // setup lookup from file
      void get_from_file(FILE *fp)
        {
        U32 num;
        fread(&num, sizeof(num), 1, fp);      // Write out number of valid array entries
        for (U32 i=0; i<num; i++)
```

```
        {
        verify r;
        r.get_from_file(fp);
        tptr->insert(r);
        }
      }
    //
    // Data Representation
    private:
      WCValSortedVector<verify> *tptr;          // Pointer to sorted list for lookup values
    };
    class field;
    extern lookup *alloc_lookup_structs(U32 type, field *f);

    ////////////////////////////////////////////////////////////////////////
    //   PA.HPP
    ////////////////////////////////////////////////////////////////////////

    #define ANLZ_CUR          90
    #define ANLZ_NEXT         92
    #define ANLZ_PREV         91
    #define ANLZ_QUIT         95
    #define ANLZ_RUN          94
    #define ANLZ_TOTAL        93
    #define FIELDCHANNEL_NEXT    605
    #define FIELDCHANNEL_PREV    604
    #define FIELD_ARRAY       315
    #define FIELD_BITSWAP     318
    #define FIELD_BYTESWAP    319
    #define FIELD_COUNTBITS   313
    #define FIELD_FILTER      320
    #define FIELD_FORMAT      311
    #define FIELD_HDRLEN      306
    #define FIELD_LEN         309
    #define FIELD_NAME        307
    #define FIELD_NEXT        310
    #define FIELD_NEXTPTL     304
```

```
#define FIELD_NEXT_BUTTON     303
#define FIELD_NUMBER          301
#define FIELD_OFFSET          308
#define FIELD_ONEVALUE        314
#define FIELD_PKTLEN          305
#define FIELD_PREVIOUS_BUTTON 302
#define FIELD_STATS           312
#define FIELD_TREE            317
#define FIELD_WORDSWAP        321
#define FILTERCHANNEL_FILTERS 602
#define FILTERCHANNEL_NUMBER  600
#define FILTERCHANNEL_TOTAL   601
#define FILTER_ALL            520
#define FILTER_ARRAY          506
#define FILTER_CHANNEL        534
#define FILTER_DECIMAL        512
#define FILTER_DELETE         516
#define FILTER_EVEN           519
#define FILTER_HEX            511
#define FILTER_INDEX          531
#define FILTER_INSERT         515
#define FILTER_MAXEDIT        522
#define FILTER_MAXLB          525
#define FILTER_MINEDIT        521
#define FILTER_MINLB          524
#define FILTER_MODIFY         517
#define FILTER_NAME           532
#define FILTER_NEXT           535
#define FILTER_ODD            518
#define FILTER_ONEVALUE       505
#define FILTER_PREV           536
#define FILTER_SIGNED         513
#define FILTER_TREE           508
#define FILTER_UNSIGNED       514
#define NP_ALL                409
#define NP_APPEND             414
#define NP_ARRAY              435
```

```
#define NP_DECIMAL        417
#define NP_DELETE         416
#define NP_EVEN           408
#define NP_HEX            418
#define NP_INSERT         414
#define NP_MAXEDIT        402
#define NP_MAXLIST        406
#define NP_MINEDIT        401
#define NP_MINLIST        405
#define NP_MODIFY         429
#define NP_NEXTIDXEDIT    403
#define NP_NEXTIDXLIST    410
#define NP_NEXTPTLEDIT    404
#define NP_NEXTPTLLIST    411
#define NP_ODD            407
#define NP_ONEVALUE       434
#define NP_PREPEND        415
#define NP_PTLS           413
#define NP_RANGENAMEEDIT  431
#define NP_RANGENAMELIST  430
#define NP_SIGNED         419
#define NP_TREE           437
#define NP_UNSIGNED       420
#define NP_USE            412
#define PTL_ADD           1001
#define PTL_AFTER         1003
#define PTL_BITLEN        203
#define PTL_DELETE        1002
#define PTL_DSPWIDTH      205
#define PTL_INSERT        1004
#define PTL_NAME          201
#define PTL_NUMFIELDS     204
#define PTL_OPTNAME       202
#define PTL_OPTSAVAIL     206

////////////////////////////////////////////////////////////////////////
//    PCOLS.CPP
```

```
////////////////////////////////////////////////////////////////////

#include "gen.hpp"


static protocols DataPtl, PadPtl;
protocol *DataPtlPtr, *PadPtlPtr;  // PUT THIS IN setup_protocols()
WCValSortedVector<protocols> *ProtocolList;          // Pointer to sorted list for lookup values

protocol *base_protocol(S8 *ProtocolName)
{
for (U32 i=0; i<ProtocolList->entries(); i++)
  if (strcmp(ProtocolName, ProtocolList->operator[](i).prot()->filename()) == 0)
    return(ProtocolList->operator[](i).prot());
return(0);
}

void setup_protocols()
{
U32 number=0;
find_t fileblock;

//
// Determine number of protocol definitions that exist by counting files
//
if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
  {
  do
    {
    ++number;
    } while(_dos_findnext(&fileblock) == 0);
  }

//
// Setup a vector twice the size of the current number with number for the grow size
//
ProtocolList = new WCValSortedVector<protocols>(number*2, number);
```

```
//
// Open each protocol definition file; create a protocol definition in memory; insert it into the vector
//
if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
  {
  do
    {
    FILE *fp = fopen(fileblock.name, "rb");
    if ((fp != 0) && (fileblock.size > 56))
      {
      U32 tmp;
      fread(&tmp,    sizeof(tmp),    1,  fp); // Get length of protocol name
//
      S8 *pname = (S8 *)malloc(tmp);
      fread(pname,    tmp,          1,  fp); // Get file name
      fclose(fp);
      protocol *proto = new protocol(pname, fileblock.name);
//
      protocols p(pname, proto);
      free(pname);
      ProtocolList->insert(p);
      }
    } while(_dos_findnext(&fileblock) == 0);
  }

//
// ? Do we want to do something about duplicate protocol names here?
//
//      for (U32 ii=0, jj = 1; ii<ProtocolList->entries(); ii++, jj++)
//        {
//        if ((ii != jj) && (strcmp( ProtocolList->operator[](ii).prot()->pname(),
//          ProtocolList->operator[](jj).prot()->pname()) == 0))
//        ProtocolList->removeAt(jj);
//        }


//
```

```
// Open each protocol definition file; create all required data structures
//
if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
  {
  do
    {
    FILE *fp = fopen(fileblock.name, "rb");
    if ((fp != 0) && (fileblock.size > 56))
      {
      protocols p(fp), t;   // Reads name_length and name
      if (ProtocolList->find(p, t) != FALSE)
        t.prot()->get_from_file(fp);
      fclose(fp);
      }
    } while(_dos_findnext(&fileblock) == 0);
  }

protocols data("DATA", 0), pad("PAD", 0);          // PUT THIS IN setup_protocols()
ProtocolList->find(data, DataPtl);                 // PUT THIS IN setup_protocols()
ProtocolList->find(pad, PadPtl);                   // PUT THIS IN setup_protocols()
DataPtlPtr = DataPtl.prot();                       // PUT THIS IN setup_protocols()
PadPtlPtr = PadPtl.prot();                         // PUT THIS IN setup_protocols()

//setup_criteria();              // filter channels are setup - tie criteria to fields
}

protocol *setup_newprotocol(S8* szFileName)
{
U32 number=0;
find_t fileblock;
S8 *pname = (S8 *)malloc(50);
//
strcpy(pname,szFileName);
strcpy(strrchr(pname, '.'), "\0 ");
strcpy(pname, (strrchr(pname, '\\')+1));
strcpy(szFileName, (strrchr(szFileName, '\\')+1));
//
```

```
// See if there are any protocols
//
if (_dos_findfirst("*.pdf", _A_NORMAL, &fileblock) == 0)
  {
  do
    {
    ++number;
    } while(_dos_findnext(&fileblock) == 0);
  }

//
// Setup a vector if there are no .pdfs
//
if (number == 0)
ProtocolList = new WCValSortedVector<protocols>(10, 5);  // use some other define/default

protocol   *proto = new protocol(pname, szFileName);
protocols  p(pname, proto);
field      *tmpfs = new field[1];
field      f(0,0,24);

tmpfs[0] = f;
proto->fieldptr(tmpfs);
proto->numbits(tmpfs->bitlen());
proto->numfields(1);

free(pname);
ProtocolList->insert(p);
FILE *fp = fopen(szFileName, "wb");
if (fp != 0)
  {
  p.prot()->out_to_file();
  fclose(fp);
  p.prot()->clear_out_flag();
  return(proto);
  }
return(FALSE);
```

```
}

//////////////////////////////////////////////////////////////////
//    ROUTE.HPP
//////////////////////////////////////////////////////////////////

enum {NUM_INTFS=256};
enum InterfaceTypes {NOINTF, FDDI, TOKEN_RING, ETHERNET};
enum FrameTypes    {E802_3=4, E80D5, Ev2, TR_SNAP};

class  RouteTableEntry
{
public:
//
// Constructors
  RouteTableEntry() { memset(DstNetAddr, 0, sizeof(RouteTableEntry)); }
//
// Used for creating actual Table Entry
  RouteTableEntry(S8 *Dna, S8 *DnM, U32 Ft, U32 Di,
            U32 Mhl, S8 *Mh, U32 MinSz, U32 MaxSz)
    : DstFrameType((U16)Ft), DstInterface((U16)Di), MacHdrLen((U16)Mhl),
     DataLen(0), DataPtr(0), MinLen((U16)MinSz), MaxLen((U16)MaxSz)
    {
    DstNetAddr[0] = ((U32 *)Dna)[0] & ((U32 *)DnM)[0];
    DstNetAddr[1] = ((U32 *)Dna)[1] & ((U32 *)DnM)[1];
    DstNetAddr[2] = ((U32 *)Dna)[2] & ((U32 *)DnM)[2];
    if (MacHdrLen != 0)
      {
      MacHdr = new S8[MacHdrLen];
      memcpy(MacHdr, Mh, MacHdrLen);
      }
    }
//
// Destructor
  ~RouteTableEntry() { if (MacHdr != 0) delete []MacHdr; }
//
// Overloaded == operator...for WCPtrSortedVector<RouteTableEntry> class
```

```
int operator==(const RouteTableEntry& rte) const
  { return(DstNetAddr[0] == rte.DstNetAddr[0] &&
       DstNetAddr[1] == rte.DstNetAddr[1] &&
       DstNetAddr[2] == rte.DstNetAddr[2]); }
//
// Overloaded < operator
  int operator <(const RouteTableEntry &rte) const
    { return(wordswap(DstNetAddr[0]) < wordswap(rte.DstNetAddr[0]) ||
         wordswap(DstNetAddr[1]) < wordswap(rte.DstNetAddr[1]) ||
         wordswap(DstNetAddr[2]) < wordswap(rte.DstNetAddr[2])); }
//
// UpdateRteData()
//   Function:  Update Data pointer and Length
//          if destination is 802.3 setup length field
//
  void UpdateRteData()
    {
    DataPtr = ParsePtr;
    DataLen = (U16)(FrameLen - (ParsePtr - FramePtr)*8);
    if (DstFrameType == E802_3)   // Compute and Update 802.3 Length Field
      ((U16 *)MacHdr)[6] =
          byteswap((U16)(((U32)DataLen + (U32)MacHdrLen)/8 - 14));
    }
//
// Used for updating lookup address only
  UpdateAddr(S8 *Dna, S8 *DnM)
    {
    DstNetAddr[0] = ((U32 *)Dna)[0] & ((U32 *)DnM)[0];
    DstNetAddr[1] = ((U32 *)Dna)[1] & ((U32 *)DnM)[1];
    DstNetAddr[2] = ((U32 *)Dna)[2] & ((U32 *)DnM)[2];
    }
//
// RouteTableEntry::RouteFrame()
//   Function:    Setup Data pointer and Length
//
//   Returns:TRUE - If Frame is ready to transmit
//         FALSE - If Frame is too small/large for device
```

```
//          FALSE - If Destination is Internal (DstInterface == NUM_INTFS)
//
  U32 RouteFrame()
    {
    if (DstInterface == NUM_INTFS)          return(FALSE);
    U32 TotalLen = DataLen + MacHdrLen;
    if (TotalLen < MinLen || TotalLen > MaxLen)   return(FALSE);
    return(TRUE);
    }
//
private:
    U32 DstNetAddr[3]; // This Value is used to look up a route
    U16 DstFrameType;  // 802.3, Ethernet v2, "80D5", FDDI, Token Ring, ...
    U16 DstInterface;  // Interface to transmit this frame on
    U16 MacHdrLen;     // Length of MAC header and encapsulation
    U16 DataLen;       // Length of data less MAC header and encapsulation
    U16 MinLen;        // Allow selectable minimum size per device
    U16 MaxLen;        // Allow selectable maximum size per device
    S8  *MacHdr;       // Pointer to MAC header and encapsulation
    S8  *DataPtr;      // Pointer to data to transmit for this station
};


static RouteTableEntry TmpRte;

class RouteTable
{
public:
//
// Constructor...Route tables are not allowed to resize
  RouteTable(U32 AddrSz, U32 TableSize=8192)
    { // Use Address Size to generate mask...allocate fixed size Route Table
    for(U32 i=0; i<AddrSz; i++)NetMask[i] = 0xff;
    for(    ;i<12;   i++)NetMask[i] = 0;
    Table = new WCPtrSortedVector<RouteTableEntry>(TableSize, 0);
    }
//
```

```
    // Destructor
       ~RouteTable() {delete Table;}
    //
    // Remove Entry from Routing Table...returns NULL if entry not found
       RouteTableEntry *remove(RouteTableEntry *rte) {return(Table->remove(rte));}
    //
    // Insert Entry into Routing Table...Returns NULL if unsuccessful
       int insert(RouteTableEntry *rte) {return(Table->insert(rte));}
    //
    // Lookup Address in Route Table
       RouteTableEntry *RouteFrame()
          {
          RouteTableEntry *RtePtr;
          TmpRte.UpdateAddr(ParsePtr + ProtoParseLen/8, NetMask);
          if ((RtePtr = Table->find(&TmpRte)) != 0)
             RtePtr->UpdateRteData();
          return(RtePtr);
          }
    //
    private:
       S8 NetMask[12];                    // Address Mask
       WCPtrSortedVector<RouteTableEntry> *Table; // Pointer to Routing Table
    };
    //
    extern RouteTableEntry *RtePtr;
    -

    ////////////////////////////////////////////////////////////////////////
    //    STAT.HPP
    ////////////////////////////////////////////////////////////////////////

    //
    //
    // Statistics Base Class Definition
    //    Virtual member functions
    //       collect()  -  accumulate statistics
    //       clear()    -  clear statistics counter
```

```
//
enum stattypes {  NOSTATS,
                  SUM,     INC,   SUMINC,
                  IDXSUM,  IDXINC, IDXSUMINC,
                  HASHSUM,  HASHINC, HASHSUMINC,
                  IDXINCSUM, HASHINCSUM,  CSUMINC };
class stats
{
public:
  stats(S8 *name=0): stat_name(0)
    {
    if (name != 0)
      {
      stat_name = new S8[strlen(name)+1];
      strcpy(stat_name, name);
      }
    }
  virtual ~stats() {if (stat_name != 0) delete stat_name;}
  virtual void collect(U32 value, U32 flen=0) {value=flen;}// default behavior
  virtual void clear() {}                    // default behavior
  virtual void clear (U32 idx) { idx = idx; }        // default behavior
  virtual U32 get_type() { return(NOSTATS); }

protected:
  S8  *stat_name;      // Pointer to user supplied statistic name

private:
};

//////////////////////////////////////////////////////////////////////
// Derived Summation Statistics Class Definition
//   For use where the contents of a protocol field need to be accumulated
//
//   Virtual member functions
//      collect(U32 value) - adds value to summation variable
//      clear()        - sets summation variable to 0
//////////////////////////////////////////////////////////////////////
```

```
class sum_stats: public stats
{
public:
//
// Constructor
  sum_stats(S8 *name): stats(name), sum(0,0)   {}
//
// Destructor
  ~sum_stats()   {}
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// Add field value to sum of previous values
  void collect(U32 value, U32 flen=0) { flen = flen; sum += value; }
//
// Clear statistic
  void clear()        { sum = 0; }
//
// return Statistic type
  U32 get_type() const   { return(SUM); }
//
// Data Representation
private:
  U64 sum;            // Pointer to U64 variable for summing
};

////////////////////////////////////////////////////////////////////////
// Derived Increment Statistics Class Definition
//   For use where the occurances of a protocol field need to be counted
//
//   Virtual member functions
//      collect(U32 value) - increments count variable
//      clear()          - sets count variable to 0
////////////////////////////////////////////////////////////////////////
class inc_stats: public stats
```

```
{
public:
//
// Constructor
  inc_stats(S8 *name): stats(name), count(0,0) {}
//
// Destructor
  ~inc_stats()  {}
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// Increment number of times this field parsed
  void collect(U32 value, U32 flen=0) { flen=flen;  value=value ; ++count; }
//
// Clear statistic
  void clear()  { count = 0; }
//
// return statistic type
  U32 get_type()  { return(INC); }
//
// Data Representation
private:
  U64 count;          // Pointer to U64 variable for counting
};

////////////////////////////////////////////////////////////////////////
// Derived Summation Statistics Class Definition
//   For use where the contents of a protocol field need to be accumulated
//   as well as a count mainatined
//
//   Virtual member functions
//     collect(U32 value) - adds field content value to summation
//                          variable and increments count variable
//     clear()          - sets summation and count variables to 0
////////////////////////////////////////////////////////////////////////
```

```
class suminc_stats: public stats
{
public:
//
// Constructor
  suminc_stats(S8 *name): stats(name), sum(0,0), count(0,0) {}
//
// Destructor
  ~suminc_stats() {}
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// Add field value to sum of previous values and increment counter
  void collect(U32 value, U32 flen=0) { flen=flen; sum += value; ++count; }
//
// Clear statistic
  void clear()   { sum = count = 0; }
//
// return statistic type
  U32 get_type()  { return(SUMINC); }
//
// Data Representation
private:
  U64 sum;          // Pointer to U64 variable for summing
  U64 count;        // Pointer to U64 variable for counting
};

////////////////////////////////////////////////////////////////////////
// Derived Indexed Summation Statistics Class Definition
//   For use where the contents of a protocol field need to be accumulated
//   in an array indexed by the value itself
//
//   Virtual member functions
//      collect(U32 value) - adds field content value to counter and
//                           increments array indexed by value
```

```
//      clear()          - sets each array element to 0
//      clear(U32 idx)   - sets array[idx] to 0
////////////////////////////////////////////////////////////////////
class idxsum_stats: public stats
{
public:
//
// Constructor
  idxsum_stats(S8 *name, U32 maxsz): stats(name), size(maxsz)
    { array = new U64[maxsz+1]; }
//
// Destructor
  ~idxsum_stats()      { delete []array; }
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// Index using field value and add field value to sum of previous values
  void collect(U32 val, U32 flen=0) { flen=0; array[min(val, size)] += val; }
//
// Clear statistic
  void clear()         { memset(array, 0, size*sizeof(U64)); }
  void clear(U32 idx)  { if (idx <= size) array[idx] = 0; }
//
// return statistic type
  U32 get_type()       { return(IDXSUM); }
//
// Data Representation
private:
  U32 size;          // Maximum possible statistic value + 1
  U64 *array;        // Space for (size+1) elements
};

////////////////////////////////////////////////////////////////////
// Derived Index Increment Statistics Class Definition
//   For use where occurances of a protocol field need to be counted
```

```
//
//   Virtual member functions
//       collect(U32 value) -  adds 1 to counter
//       clear()          -  sets counter value to 0
////////////////////////////////////////////////////////////////////////
class idxinc_stats: public stats
{
public:
//
// Constructor
  idxinc_stats(S8 *name, U32 maxsz): stats(name), size(maxsz)
    { array = new U64[maxsz+1]; }
//
// Destructor
  ~idxinc_stats()    { delete []array; }
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// Index on field value and increment counter
  void collect(U32 value, U32 flen=0){ flen=flen; ++array[min(value, size)]; }
//
// Clear statistic
  void clear()         { memset(array, 0, size*sizeof(U64)); }
  void clear(U32 idx)    { if (idx <= size) array[idx] = 0; }
//
// return statistic type
  U32 get_type()       { return(IDXINC); }
//
// Data representation
private:
  U32 size;           // Maximum possible statistic value + 1
  U64 *array;          // Space for (size+1) elements
};
////////////////////////////////////////////////////////////////////////
// Derived Index Sum and Increment Statistics Class Definition
```

```
//    For use where occurances of a protocol field need to be counted in an
//    array indexed by the value itself, and the contents of a protocol field
//    need to be accumulated
//
//    Virtual member functions
//      collect(U32 value) -  adds field content value to counter and
//                            increments array indexed by value
//                            and adds 1 to counter
//      clear()            -  sets counter value and entire array to 0
//      clear(U32 idx)     -  sets array[idx] to 0 and
//                            sets counter value to 0
////////////////////////////////////////////////////////////////////////////
class idxsuminc_stats: public stats
{
public:
//
// Constuctor
  idxsuminc_stats(S8 *name, U32 maxsz):stats(name), size(maxsz)
    { asum = new U64[maxsz+1]; ainc = new U64[maxsz+1]; }
//
// Destructor
  ~idxsuminc_stats() { delete []asum; delete []ainc; }
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// Index on field value, increment counter, add value to sum of previous values
  void collect(U32 val, U32 flen=0)
    { flen=flen; val = min(val, size); asum[val] += val; ++ainc[val]; }
//
// Clear statistic
  void clear()
  { memset(asum, 0, size*sizeof(U64)); memset(ainc, 0, size*sizeof(U64)); }
  void clear(U32 idx) { if (idx <= size) asum[idx] = ainc[idx] = 0; }
//
// return statistic type
```

```
    U32 get_type()  { return(IDXSUMINC); }
  //
  // Data Representation
  private:
    U32 size;          // Maximum possible statistic value + 1
    U64 *asum;         // Space for (size+1) elements
    U64 *ainc;         // Space for (size+1) elements
  };

  ////////////////////////////////////////////////////////////////////////
  // Derived Sum Hash Table Entry Class Definition
  //    For use where the contents of a protocol field with many possible values needs to
  //    acumulate the total at each received value
  //
  //    Virtual member functions
  //       collect(U32 value) - adds 1 to counter
  //       clear()            - sets counter value to 0
  ////////////////////////////////////////////////////////////////////////
  inline void *u64_alloc(size_t sz)
    { sz = sz; return((void *)malloc(WCPtrHashDictItemSize(U32, U64))); }
  //
  inline void u64_dealloc(void *ptr, size_t sz) { sz = sz; delete ptr; }
  //
  class hashsum_stats: public stats
  {
  public:
    static unsigned h_fn(const U32& u) { return(u); }
  //
  // Constructor
    hashsum_stats(S8 *name, U32 hsz=WC_DEFAULT_HASH_SIZE): stats(name)
      { ht = new WCPtrHashDict<U32, U64>(h_fn, hsz, u64_alloc, u64_dealloc); }
  //
  // Destructor
    ~hashsum_stats()   { delete ht; }
  //
  // Input/Output Routines
    void out_to_file(FILE *fp) const;
```

```
    void get_from_file(FILE *fp);
  //
  // use value to hash into table and add value to sum of previous values
    void collect(U32 value, U32 flen=0)
      {
      flen = flen;
      U64 *u;
      U32 *uptr, * &val = uptr;
      if ((u = ht->findKeyAndValue(&value, val)) == 0)
        ht->insert(&value, u = new U64);
      *u += value;
      }
  //
    static void zero(U32 *key, U64 *value, void *data)
      { key = key; data = data; value->clear(); }
  //
  // Clear statistic
    void clear()   { ht->forAll(zero, 0); }
  //
  // return statistic type
    U32 get_type()  { return(HASHSUM); }
  //
  // Data Representation
  private:
    WCPtrHashDict<U32, U64> *ht;      // Pointer to hash table for U64 values
  };

  ////////////////////////////////////////////////////////////////////////
  // Derived Increment Hash Table Entry Class Definition
  //   For use where occurances of a protocol field need to be counted and
  //   the contents are sparsely distributed
  //
  //   Virtual member functions
  //      collect(U32 value) -  adds 1 to counter
  //      clear()          -  sets counter value to 0
  ////////////////////////////////////////////////////////////////////////
  class hashinc_stats: public stats
```

```
{
public:
  static unsigned h_fn(const U32& u) { return(u); }
//
// Constructor
  hashinc_stats(S8 *name, U32 hsz=WC_DEFAULT_HASH_SIZE): stats(name)
    { ht = new WCPtrHashDict<U32, U64>(h_fn, hsz, u64_alloc, u64_dealloc); }
//
// Destructor
  ~hashinc_stats()  { delete ht; }
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// use value to hash into table and increment counter
  void collect(U32 value, U32 flen=0)
    {
    U64 *u;
    U32 *uptr, * &val = uptr;
    flen = flen;
    if ((u = ht->findKeyAndValue(&value, val)) == 0)
      ht->insert(&value, u = new U64);
    ++(*u);
    }
//
  static void zero(U32 *key, U64 *value, void *data)
    { key = key; data = data; value->clear(); }
//
// Clear statistic
  void clear()  { ht->forAll(zero, 0); }
//
// return statistic type
  U32 get_type() { return(HASHINC); }
//
// Data Representation
private:
```

```
    WCPtrHashDict<U32, U64> *ht;      // Pointer to hash table for U64 values
};

////////////////////////////////////////////////////////////////////////
// Derived Sum Hash Table Entry Class Definition
//    For use where the contents of a protocol field with many possible
//    values needs to accumulate the total at each received value
//
//    Virtual member functions
//       collect(U32 value) - adds 1 to counter
//       clear()            - sets counter value to 0
////////////////////////////////////////////////////////////////////////
class suminc
{
public:
//
// Constructor
  suminc(): count(0), summ(0) {}
//
// Destructor
  ~suminc()        {}
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// increment counter
  inc()            { ++count; }
//
// add value to sum of previous values
  sum(U32 value)   { summ += value; }
//
// increment counter and add value to sum of previous values
  incsum(U32 value)  { ++count; summ += value; }
//
// Clear statistic
  void clear()     { count = summ = 0; }
```

```
    //
    // Data Representation
    private:
      U64 count, summ;  // Pointer to counter and summation statistic variables
    };

    ////////////////////////////////////////////////////////////////////////
    //
    ////////////////////////////////////////////////////////////////////////
    class hashsuminc_stats: public stats
    {
    public:
      static unsigned h_fn(const U32& u) { return(u); }
      static void *si_alloc(size_t sz)
        { sz = sz; return((void *)malloc(WCPtrHashDictItemSize(U32, suminc))); }
      static void si_dealloc(void *ptr, size_t sz) { sz = sz; delete ptr; }
    //
    // Constructor
      hashsuminc_stats(S8 *name, U32 sz=WC_DEFAULT_HASH_SIZE): stats(name)
        { ht = new WCPtrHashDict<U32, suminc>(h_fn, sz, si_alloc, si_dealloc); }
    //
    // Destructor
      ~hashsuminc_stats()    { delete ht; }
    //
    // Input/Output Routines
      void out_to_file(FILE *fp) const;
      void get_from_file(FILE *fp);
    //
    // use value to hash into table, increment counter and sum value
      void collect(U32 value, U32 flen=0)
        {
        suminc *u;
        U32 *uptr, * &val = uptr;
        flen = flen;
        if ((u = ht->findKeyAndValue(&value, val)) == 0)
          ht->insert(&value, u = new suminc);
        u->incsum(value);
```

```
    }
  //
    static void zero(U32 *key, suminc *value, void *data)
      { key = key; data = data; value->clear(); }
  //
  // Clear statistic
    void clear()   { ht->forAll(zero, 0); }
  //
  // return statistic type
    U32 get_type()  { return(HASHSUMINC); }
  //
  // Data Representation
  private:
    WCPtrHashDict<U32, suminc>  *ht;      // Pointer to hash table<suminc>
  };
  ////////////////////////////////////////////////////////////////////
  // Derived Increment Index Sum and Sum length field Statistics Class Definition
  //   For use where the occurances of a protocol field need to be counted in
  //   an array indexed by the value itself, and the supplied protocol length
  //   needs to be accumulated
  //
  //   Virtual member functions
  //     collect(U32 value) - adds field content value to counter and
  //                          increments array indexed by value
  //                          and adds 1 to counter
  //     clear()            - sets counter value and entire array to 0
  //     clear(U32 idx)     - sets array[idx] to 0 and
  //                          sets counter value to 0
  ////////////////////////////////////////////////////////////////////
  class idxincsum_stats: public stats
  {
  public:
  //
  // Constuctor
    idxincsum_stats(S8 *name, U32 maxsz): stats(name), size(maxsz)
      { asum = new U64[maxsz+1]; ainc = new U64[maxsz+1]; }
  //
```

```
    // Destructor
      ~idxincsum_stats() { delete []asum; delete []ainc; }
    //
    // Input/Output Routines
      void out_to_file(FILE *fp) const;
      void get_from_file(FILE *fp);
    //
    // index on value, increment counter,
      void collect(U32 value, U32 flen=0)
        {
        value = min(value, size);
        ++ainc[value];
        asum[value] += flen;
        }
    //
    // clear statistics
      void clear()
        { memset(asum, 0, size*sizeof(U64));memset(ainc, 0, size*sizeof(U64)); }
    //
      void clear(U32 idx) { if (idx <= size) asum[idx] = ainc[idx] = 0; }
    //
    // return statistic type
      U32 get_type() { return(IDXINCSUM); }
    //
    // Data Representation
    private:
      U32 size;          // Maximum possible statistic value + 1
      U64 *asum;         // Space for (size+1) elements
      U64 *ainc;         // Space for (size+1) elements
    };

    ////////////////////////////////////////////////////////////////////////
    // Derived Increment Hash Table Entry and Sum Length Field Class Definition
    //   For use where the number of occurances of a protocol field must be
    //   counted and associated packet length summed
    //
    //   Virtual member functions
```

```
//      collect(U32 value) - adds 1 to counter
//      clear()          - sets counter value to 0
////////////////////////////////////////////////////////////////////////
class hashincsum_stats: public stats
{
public:
//
  static unsigned h_fn(const U32& u) { return(u); }
  static void *is_alloc(size_t sz)
    { sz = sz; return((void *)malloc(WCPtrHashDictItemSize(U32, suminc))); }
  static void is_dealloc(void *ptr, size_t sz) { sz = sz; delete ptr; }
//
// Constructor
  hashincsum_stats(S8 *name, U32 sz=WC_DEFAULT_HASH_SIZE): stats(name)
    { ht = new WCPtrHashDict<U32, suminc>(h_fn, sz, is_alloc, is_dealloc); }
//
// Destructor
  ~hashincsum_stats()    { delete ht; }
//
// Input/Output Routines
  void out_to_file(FILE *fp) const;
  void get_from_file(FILE *fp);
//
// use value to hash into table, increment counter, sum value
  void collect(U32 value, U32 flen=0) // Hash, sum and increment entry values
    {
    suminc *u;
    U32 *uptr, * &val = uptr;
    if ((u = ht->findKeyAndValue(&value, val)) == 0)
      ht->insert(&value, u = new suminc);
    u->inc();
    u->sum(flen);
    }
//
  static void zero(U32 *key, suminc *value, void *data)
            { key = key; data = data; value->clear(); }
//
```

```
    // Clear statistic
      void clear()   { ht->forAll(zero, 0); }
    //
    // return statistic type
      U32 get_type() { return(HASHINCSUM); }
    //
    // Data Representation
    private:
      WCPtrHashDict<U32, suminc>  *ht;      // Pointer to hash table<suminc>
    };
    class field;
    extern stats *alloc_stat_structs(U32 type, field *f);

    ////////////////////////////////////////////////////////////////////////
    //    XMIT.HPP
    ////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////
    // Base Class definition for Varying Field Values
    //
    // Rules:::
    //        Right shift value must always be >= left shift value
    //        If Right and Left shift values are equal ... field contains LSB
    //        Field bitlength                 := 32 - right shift value
    //        Left shift value to original position  :=  right - left shift value
    ////////////////////////////////////////////////////////////////////////
    class vary
    {
    public:
    // Constructor
      vary(U32 shl, U32 shr, U32 op=0)
        {
        mask = ((0xffffffff << shr) >> shl);
        notmask = ~mask;
        operand = ((op << shr) >> shl);
        if (shl == shr)
          minvalue = 0;
```

```
        else
          minvalue = (0xffffffff >> (shl + (32-shr)));
        maxvalue = (0xffffffff >> shl) & mask;
        }
    virtual ~vary() {}
  // Member Functions
    virtual U32 vary_value(U32 value)
      {// (8/10 + 7/5) ... (insts/clocks)
      return((value & notmask) | ((value + operand) & mask));
      }
  protected:
    U32 mask;
    U32 notmask;
    U32 operand;
    U32 minvalue;
    U32 maxvalue;
  private:
  };

  ////////////////////////////////////////////////////////////////////
  // Derived class for incrementing field values
  ////////////////////////////////////////////////////////////////////
  class vary_incwrap: public vary
  {
  public:
  // Constructors
    vary_incwrap(U32 shl, U32 shr):        vary(shl, shr, 1) {}
  private:
  };

  ////////////////////////////////////////////////////////////////////
  // Derived class for incrementing field values within a (min:max) range
  ////////////////////////////////////////////////////////////////////
  class vary_incwrapminmax: public vary
  {
  public:
  // Constructors
```

```
    vary_incwrapminmax(U32 shl, U32 shr):          vary(shl, shr, 1) {}
  // Member Functions
    U32 vary_value(U32 value)
      { // (8/10 + 10/11) ... (instrs/clocks) ... Total ((18 | 19)/(19 | 20))
      U32 newv = (value & mask);
      if (newv == maxvalue)
        newv  = minvalue;
      else
        newv += operand;
      return((value & notmask) | newv);
      }
  private:
  };

  ////////////////////////////////////////////////////////////////////////
  // Derived class for decrementing field values
  ////////////////////////////////////////////////////////////////////////
  class vary_decwrap: public vary
  {
  public:
  // Constructors
    vary_decwrap(U32 shl, U32 shr):          vary(shl, shr, -1) {}
  private:
  };

  ////////////////////////////////////////////////////////////////////////
  // Derived class for decrementing field values within a (min:max) range
  ////////////////////////////////////////////////////////////////////////
  class vary_decwrapminmax: public vary
  {
  public:
  // Constructors
    vary_decwrapminmax(U32 shl, U32 shr):          vary(shl, shr, -1) {}
  // Member Functions
    U32 vary_value(U32 value)
      { // (8/10 + 10 | 11) ... (instrs/clocks) ... Total ((18 | 19)/(19 | 20))
      U32 newv = (value & mask);
```

```
        if (newv == minvalue)
          newv = maxvalue;
        else
          newv += operand;
        return((value & notmask) | newv);
        }
    private:
    };

    ////////////////////////////////////////////////////////////////////
    // Derived class for adding a value to a field
    ////////////////////////////////////////////////////////////////////
    class vary_addwrap: public vary
    {
    public:
    // Constructors
      vary_addwrap(U32 shl, U32 shr, U32 op=1):  vary(shl, shr, op) {}
    private:
    };

    ////////////////////////////////////////////////////////////////////
    // Derived class for adding a value to a field within a (min:max) range
    ////////////////////////////////////////////////////////////////////
    class vary_addwrapminmax: public vary
    {
    public:
    // Constructors
      vary_addwrapminmax(U32 shl, U32 shr, U32 op=1):  vary(shl, shr, op) {}
    // Member Functions
      U32 vary_value(U32 value)
        { // (10/12 + 11|12) ... (instrs/clocks) ... Total ((21|22)/(21|22))
        U32 newv = maxvalue - (value & mask);
        if (newv >= operand)
          newv += operand;
        return((value & notmask) | newv);
        }
    private:
```

```
};

////////////////////////////////////////////////////////////////////
// Derived class for subtracting a value from a field
////////////////////////////////////////////////////////////////////
class vary_subwrap: public vary
{
public:
// Constructors
  vary_subwrap(U32 shl, U32 shr, U32 op=1):  vary(shl, shr, -op) {}
private:
};

////////////////////////////////////////////////////////////////////
// Derived class for subtracting a value from a field within a (min:max) range
////////////////////////////////////////////////////////////////////
class vary_subwrapminmax: public vary
{
public:
// Constructors
  vary_subwrapminmax(U32 shl, U32 shr, U32 op=1):  vary(shl, shr, -op) {}
// Member Functions
  U32 vary_valueminmax(U32 value)
    { // (10/12 + 11|12) ... (instrs/clocks) ... Total ((21|22)/(21|22))
    U32 newv = (value & mask) - minvalue;
    if (newv >= operand)
      newv += operand;
    return((value & notmask) | newv);
    }
private:
};

////////////////////////////////////////////////////////////////////
// Derived class for rotating left a field value
////////////////////////////////////////////////////////////////////
class vary_rotl: public vary
{
```

```
public:
//
// Constructor
  vary_rotl(U32 shr, U32 shl, U32 op):      vary(shl, shr) { operand = op; }
//
// rotate the field value left 1 bit
  U32 vary_value(U32 value) { return(_rotl(value, 1)); }
private:
};

////////////////////////////////////////////////////////////////////
// Derived class for rotating right a field value
////////////////////////////////////////////////////////////////////
class vary_rotr: public vary
{
public:
//
// Constructor
  vary_rotr(U32 shr, U32 shl, U32 op):      vary(shl, shr) { operand = op; }
//
// rotate the field value right 1 bit
  U32 vary_value(U32 value) { return(_rotr(value, 1)); }
private:
};

////////////////////////////////////////////////////////////////////
// Derived class for performing incremental update of IP checksum field
////////////////////////////////////////////////////////////////////
// IP checksum incremental update routine (bitswapped)
////////////////////////////////////////////////////////////////////
extern U32 IpCsumUpd(U32 value, U32 change);
#pragma aux IpCsumUpd = \
  "add   ax,cx"    \
  "cmc"            \
  "sbb   ax,0"     \
  parm   [eax] [ecx] \
  modify [eax]     \
```

```
    value  [eax]

class vary_ipcsum: public vary
{
public:
//
// Constructor
   vary_ipcsum(U32 shr=0, U32 shl=0, U32 op=0): vary(shl, shr) { operand=op; }
//
// Member Functions
   U32 vary_value(U32 value) { return(IpCsumUpd(value, operand)); }
private:
};

////////////////////////////////////////////////////////////////////////
// Derived class for performing incremental update of IPX checksum field
////////////////////////////////////////////////////////////////////////
class vary_ipxcsum: public vary
{
public:
//
// Constructor
   vary_ipxcsum(U32 shr=0, U32 shl=0, U32 op=1): vary(shl, shr) { operand=op; }
//
// Member Functions
   U32 vary_value(U32 value);
private:
};

////////////////////////////////////////////////////////////////////////
// Derived class for performing bitswapping
////////////////////////////////////////////////////////////////////////
class vary_bitswap: public vary
{
public:
//
// Constructor (op is the number of bytes to bitswap !!!)
```

```
    vary_bitswap(U32 shr=0, U32 shl=0, U32 op=1): vary(shl, shr) { operand=op; }
  //
  // Member Functions
    U32 vary_value(U32 i)
      { for (i=0; i<operand; i++) ParsePtr[i] = bs[ParsePtr[i]]; return(0); }
  private:
  };

  class vary_minmaxincwrap: public vary
  {
  public:
  // Constructors
    vary_minmaxincwrap(U32 shr, U32 shl, U32 minv=0, U32 maxv=-1)
      : vary(shl, shr, 1)
      {
      minvalue = (minv >> (shl + (31-shr))) + 1;
      maxvalue = maxv >> shl;
      }
  // Member Functions
  private:
  };

  #ifdef notdef
  class vary_minmaxdecwrap: public vary
  {
  public:
  // Constructors
    vary_minmaxdecwrap(U32 minv=0, U32 maxv=-1): vary(-1)
      { minval=minv-1; maxval=maxv; }
  // Member Functions
    U32 vary_value(U32 value, U32 shl, U32 shr);
  private:
    U32 minval;
    U32 maxval;
  };
  #endif
  →
```

Claims

1. A system for manipulating data transmitted over one or more data communications networks, said system comprising:

a logic control module capable of accessing a plurality of programmably configurable protocol descriptions stored in a memory, said programmably configurable protocol descriptions each including a protocol control record and at least one field sub-record for defining a selected portion of a network protocol to be manipulated and a plurality of rules for manipulating said portion of said protocol;

said logic control module including frame and protocol header length determining logic, statistics gathering logic, verification and error checking logic, filtering logic, next protocol determining logic, routing logic and output formatting logic for defining and controlling, based upon said programmably configurable protocol descriptions, a series of data manipulation functions to be implemented by said system.

2. A protocol analyzer comprising:

a logic control module capable of accessing a plurality of programmably configurable protocol descriptions stored in a memory, said programmably configurable protocol descriptions each including a protocol control record and at least one field sub-record for defining a selected portion of a network protocol to be analyzed and a plurality of rules for conducting an analysis of said portion of said protocol;

said logic control module including frame and protocol header length determining logic, statistics gathering logic, verification and error checking logic, filtering logic, and next protocol determining logic for defining and controlling, based upon said programmably configurable protocol descriptions, a series of parsing,

filtering, statistics gathering and display functions implemented by said protocol analyzer.

3. A machine implemented process for parsing data transmitted over a data communications network, said process comprising the steps of:

storing at least one programmably configurable protocol description in a memory, said at least one programmably configurable protocol description comprising a protocol control record and at least one field sub-record for defining a plurality of characteristics of said data transmitted over said data communications network;

retrieving said at least one protocol description from said memory; and

providing said at least one protocol description to a logic control module, said logic control module, upon receiving said at least one protocol description, being configured to parse data received from said data communications network based upon said characteristics defined by said protocol description.

4. The process of claim 3, wherein a plurality of programmably configurable protocol descriptions are stored in said memory, and wherein said programmably configurable protocol descriptions are selectively retrieved from said memory in response to selected data sequences received from said data communications network.

5. A machine implemented process for filtering data transmitted over a data communications network, said process comprising the steps of:

storing at least one programmably configurable protocol description in a memory, said at least one programmably configurable protocol description comprising a protocol control record and at least one field sub-record for defining a plurality of characteristics of said data transmitted over said data communications network;

retrieving said at least one protocol description from said memory; and

providing said at least one protocol description to a logic control module, said logic control module, upon receiving said at least one protocol description, being configured to filter data received from said data communications network based upon said characteristics defined by said protocol description.

6. The process of claim 5, wherein a plurality of programmably configurable protocol descriptions are stored in said memory, and wherein said programmably configurable protocol descriptions are selectively retrieved from said memory in response to selected data sequences received from said data communications network.

7. A machine implemented process for routing data transmitted over a data communications network, said process comprising the steps of:

storing at least one programmably configurable protocol description in a memory, said at least one programmably configurable protocol description comprising a protocol control record and at least one field sub-record for defining a plurality of characteristics of said data transmitted over said data communications network;

retrieving said at least one protocol description from said memory; and

providing said at least one protocol description to a logic control module, said logic control module, upon receiving said at least one protocol description, being configured to route data within said data communications network based upon said characteristics defined by said protocol description.

8. The process of claim 7, wherein a plurality of programmably configurable protocol descriptions are stored in said memory, and wherein said programmably configurable

protocol descriptions are selectively retrieved from said memory in response to selected data sequences received from said data communications network.

9. A method for parsing data transmitted over a data communications network, said method comprising the steps of:

storing in a first memory a plurality of programmably configurable protocol descriptions, said programmably configurable protocol descriptions defining a plurality of characteristics of said data transmitted over said data communications network;

storing in a second memory a program for controlling a data parsing function to be executed by a processing unit, said program including instructions for causing said processing unit to selectively retrieve at least one of said programmably configurable protocol descriptions from said first memory and to vary the execution of said data parsing function based upon said at least one retrieved protocol description;

delivering to said processing unit said program for controlling said data parsing function;

enabling said processing unit to execute said data parsing function; and

delivering to said processing unit said data transmitted over said data communications network.

10. A method for filtering data transmitted over a data communications network, said method comprising the steps of:

storing in a first memory a plurality of programmably configurable protocol descriptions, said programmably configurable protocol descriptions defining a one or more filter criteria;

storing in a second memory a program for controlling a data filtering function to be executed by a processing unit, said program including instructions for causing said

processing unit to selectively retrieve at least one of said programmably configurable protocol descriptions from said first memory and to vary the execution of said data filtering function based upon said at least one retrieved protocol description;

delivering to said processing unit said program for controlling said data filtering function;

enabling said processing unit to execute said data filtering function; and

delivering to said processing unit said data transmitted over said data communications network.

11. A method for routing data transmitted over a data communications network, said method comprising the steps of:

storing in a first memory a plurality of programmably configurable protocol descriptions, said programmably configurable protocol descriptions defining one or more rules for routing data within said data communications network;

storing in a second memory a program for controlling a data routing function to be executed by a processing unit, said program including instructions for causing said processing unit to selectively retrieve at least one of said programmably configurable protocol descriptions from said first memory and to vary the execution of said data routing function based upon said at least one retrieved protocol description;

delivering to said processing unit said program for controlling said data routing function;

enabling said processing unit to execute said data routing function; and

delivering to said processing unit said data transmitted over said data communications network.

12. A network interface system comprising:

at least one network interface unit for communicating with a data communications network;

at least one memory coupled to said network interface unit, said memory being configured to store a plurality of network data files and a plurality of programmably configurable protocol descriptions; and

a logic control module coupled to said at least one network interface unit and to said at least one memory, said logic control module having the capability to selectively retrieve one or more of said programmably configurable protocol descriptions from said memory and to implement a selected data manipulation function in a manner defined by said one or more retrieved protocol descriptions.

13. The network interface system of claim 12, wherein said logic control module comprises a microprocessor and a computer program which may be executed by said microprocessor.

14. The network interface system of claim 13, wherein said computer program comprises a C++ computer program having abstract data types defined for statistics gathering, value verification, next protocol determination, filtering, value modification, display and route determination capabilities.

15. The network interface system of claim 13, wherein each of said programmably configurable protocol descriptions comprises a protocol control record and a plurality of field data records.

16. The network interface system of claim 15, wherein each said protocol control record defines an overall structure of a network protocol.

17. The network interface system of claim 16, wherein each protocol control record comprises a plurality of field sub-records, a plurality of statistics records,

a plurality of lookup records, a plurality of expression records, a plurality of filter criteria records, and a plurality of value verification records.

18. A protocol analyzer comprising:

at least one network interface unit for communicating with a data communications network;

at least one memory coupled to said network interface unit, said memory being configured to store a plurality of network data files and a plurality of programmably configurable protocol descriptions, each programmably configurable protocol description comprising a protocol control record and a plurality of field sub-records, wherein said protocol control record defines an overall data structure of a selected data communication protocol;

a logic control module coupled to said at least one network interface unit and to said at least one memory, said logic control module having the capability to selectively retrieve one or more of said programmably configurable protocol descriptions from said memory and to execute one or more network analysis functions defined by said one or more retrieved protocol descriptions; and

at least one output device coupled to said logic control module for displaying a result of a network analysis function performed by said logic control module.

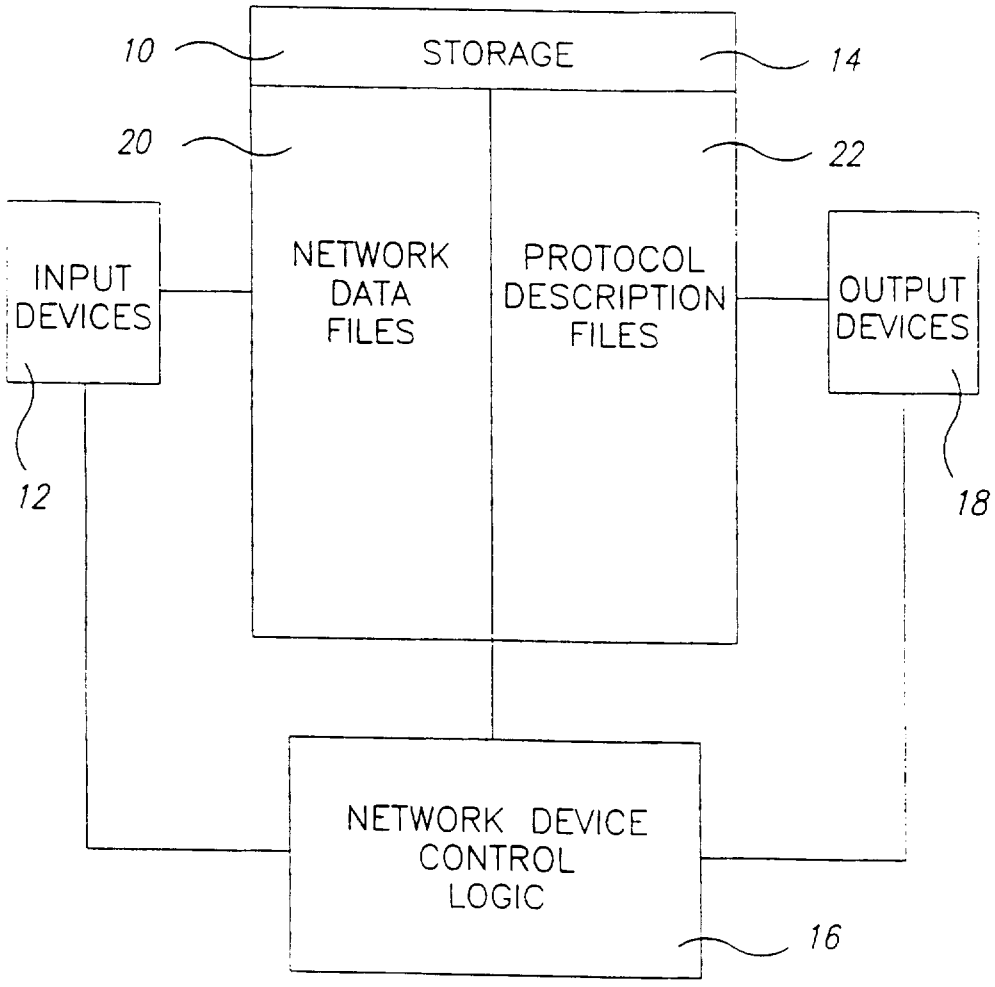


FIG. 1

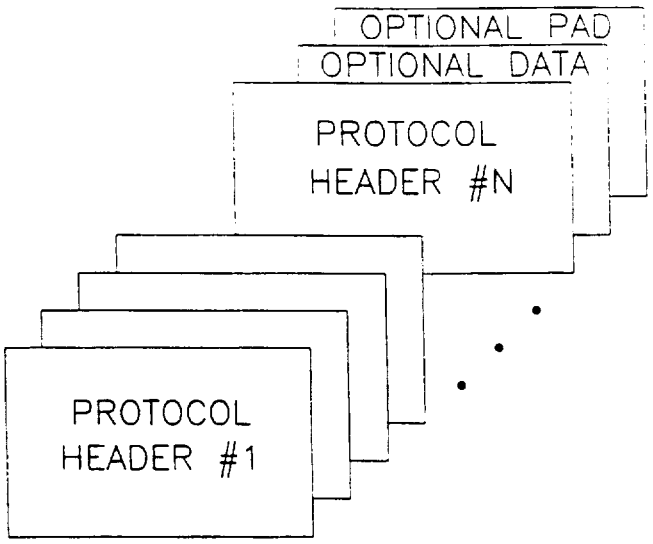


FIG. 2

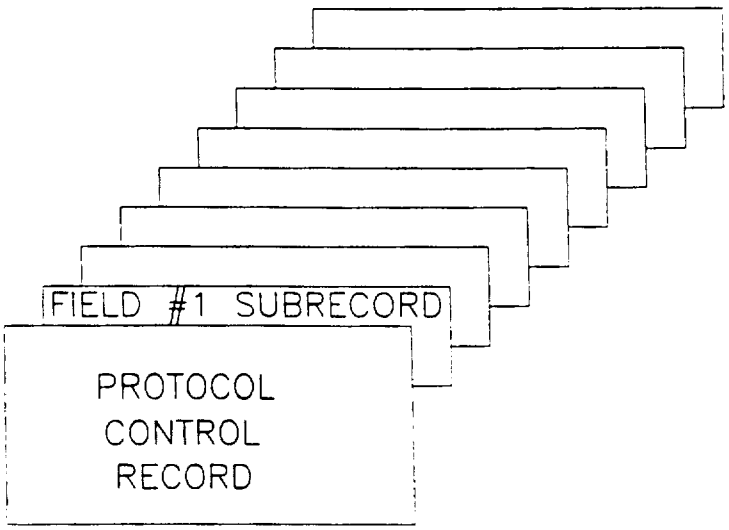


FIG. 3

Ethernet Control Record					
Protocol Name	**NumBits**	**NumFields**	**CurField**	**Fields**	**Options**
Ethernet MAC Header	112	5	0	Figure 4a	[None]

Fig. 4

Index	Field Name	Bit Offset	Bit Length	Left Shift	Right Shift	Check sum	Frame Length	Header Length	Statistics	Lookup Structure	Filter Structure	Format
0	Dst Vendor Address	0	24	0	16	0	0	0	[None]	Fig 4b	Fig 10.Idx 0	hex
1	Dst Station Address	24	24	0	16	0	0	0	[None]	[None]	Fig 10.Idx 1	hex
2	Src Vendor Address	48	24	0	16	0	0	0	[None]	Fig 4c	[None]	hex
3	Src Station Address	72	24	0	16	0	0	0	[None]	[None]	[None]	hex
4	Type	96	16	0	16	0	0	0	[None]	Fig 4d	[None]	hex

Fig. 4A

Destination Vendor Address Lookup Structure					
Protocol	**Next Index**	**Minimum**	**Maximum**	**Mask**	**Translation**
[None]	1	0x000000	0x000000	ALL	"Fast Routers, Inc."
[None]	1	0x000001	0xFFFFFF	ALL	"Unknown"

Fig. 4B

Source Vendor Address Lookup Structure

Protocol	Next Index	Minimum	Maximum	Mask	Translation
[None]	3	0x000000	0x000000	ALL	"Fast Routers, Inc."
[None]	3	0x000001	0xFFFFFF	ALL	"Unknown"

Fig. 4C

Ethernet Type Lookup Structure

Protocol	Next Index	Minimum	Maximum	Mask	Translation
[None]	5	0x0000	0x8887	ALL	"Unknown"
Figure 5	5	0x8888	0x8888	ALL	"GP"
[None]	5	0x8889	0xFFFF	ALL	"Unknown"

Fig. 4D

GP Control Record					
Protocol Name	NumBits	NumFields	CurField	Fields	Options
GP - Generic Protocol	160	11	0	Figure 5a	Figure 6

Fig. 5

Index	Field Name	Byte Offset	Bit Length	Left Shift	Right Shift	Check sum	Frame Length	Header Length	Statistics	Lookup Structure	Filter	Format
0	Version No.	0	4	0	28	0	0	0	Cnt/Index&Cnt	[None]	[None]	decimal
1	HeaderLen	0	4	4	28	0	0	32	Sum/Index&Cnt	Fig 5b	[None]	decimal
2	Frame Length	0	16	8	16	0	8	0	Sum	[None]	[None]	decimal
3	Frame Type	0	8	24	24	0	0	0	Index&Cnt	Fig 5c	Fig 10,Idx 2	hex
4	Checksum	4	16	0	16	ptr	0	0	[None]	[None]	[None]	hex
5	Control	4	8	16	24	0	0	0	[None]	[None]	[None]	bitfield
6	Hop Count	4	8	24	24	0	0	0	[None]	[None]	[None]	decimal
7	Source Socket	8	16	0	16	0	0	0	[None]	Fig 5d	[None]	hex
8	Destination Socket	8	16	16	16	0	0	0	[None]	Fig 5e	[None]	hex
9	Source Address	12	32	0	0	0	0	0	[None]	[None]	[None]	hex
10	Destination Address	16	32	0	0	0	0	0	[None]	[None]	[None]	hex

Fig. 5A

Header Length Lookup Structure

Protocol	Next Index	Minimum	Maximum	Mask	Translation
[None]	11	0x0	0x4	ALL	"Invalid Length"
[None]	2	0x5	0xF	ALL	"bytes"

Fig. 5B

Frame Type Next Protocol Structure

Protocol	Next Index	Minimum	Maximum	Mask	Translation
[None]	4	0x00	0x00	ALL	"Illegal Protocol"
GP1	4	0x01	0x01	ALL	"GP1"
GP2	5	0x02	0x02	ALL	"GP2"
[None]	4	0x03	0xFF	ALL	"Illegal Protocol"

Fig. 5C

Source Socket Next Protocol Structure

Protocol	Next Index	Minimum	Maximum	Mask	Translation
[None]	8	0x0000	0x0142	ALL	"Unknown Protocol"
GP3	8	0x0143	0x018F	ODD	"GP3"
GP4	8	0x0143	0x018F	EVEN	"GP4"
[None]	8	0x0190	0xFFFF	ALL	"Illegal Protocol"

Fig. 5D

Destination Socket Next Protocol Structure

Protocol	Next Index	Minimum	Maximum	Mask	Translation
[None]	9	0x0000	0x0142	ALL	"Unknown Protocol"
GP3	9	0x0143	0x018F	ODD	"GP3"
GP4	9	0x0143	0x018F	EVEN	"GP4"
[None]	9	0x0190	0xFFFF	ALL	"Illegal Protocol"

Fig. 5E

GP Master Option Control Record					
Protocol Name	NumBits	NumFields	CurField	Fields	Options
GP Master Option	0	1	0	Figure 6a	[None]

Fig. 6

Index	Field Name	Bit Offset	Bit Length	Left Shift	Right Shift	Check sum	Frame Length	Header Length	Statistics	Lookup Structure	Filter	Format
0	[None]	0	0	0	24	0	0	0	[None]	Figure 6b	[None]	[None]

Fig. 6A

Vendor Address Lookup Structure					
Protocol	Next Index	Minimum	Maximum	Mask	Translation
Figure 7	1	0x00	0x00	ALL	"GP EOL Option"
Figure 8	1	0x01	0x01	ALL	"GP NoOp Option"
Figure 9	1	0x02	0x02	ALL	"GP MaxSize Option"
[None]	1	0x03	0xFF	ALL	"Unknown Option"

Fig. 6B

GP EOL Option Control Record					
Protocol Name	NumBits	NumFields	CurField	Fields	Options
GP End of List Option	8	1	0	Figure 7a	[None]

Fig. 7

Index	Field Name	Bit Offset	Bit Length	Left Shift	Right Shift	Check sum	Frame Length	Header Length	Statistics	Lookup Structure	Filter	Format
0	EOL	0	8	0	24	0	0	0	[None]	Figure 7b	[None]	hex

Fig. 7A

EOL Lookup Structure				
Protocol	Next Index	Minimum	Maximum	Mask	Translation
Figure 6	1	0x00	0x00	ALL	"EOL"

Fig. 7B

GP NoOp Option Control Record					
Protocol Name	NumBits	NumFields	CurField	Fields	Options
GP NoOp Option	8	1	0	Figure 8a	[None]

Fig. 8

Index	Field Name	Bit Offset	Bit Length	Left Shift	Right Shift	Check sum	Frame Length	Header Length	Statistics	Lookup Structure	Filter	Format
0	NoOp	0	8	0	24	0	0	0	[None]	Figure 8b	[None]	hex

Fig. 8A

NoOp Lookup Structure				
Protocol	Next Index	Minimum	Maximum	Mask	Translation
Figure 6	1	0x01	0x01	ALL	"NoOp"

Fig. 8B

GP MinMaxSize Option Control Record					
Protocol Name	**NumBits**	**NumFields**	**CurField**	**Fields**	**Options**
MaxSize Option	48	4	0	Figure 9a	[None]

Fig. 9

Index	Field Name	Bit Offset	Bit Length	Left Shift	Right Shift	Check sum	Frame Length	Header Length	Statistics	Lookup Structure	Filter	Format
0	Option Type	0	8	0	24	0	0	0	[None]	Figure 9b	[None]	hex
1	Option Length	0	8	8	24	0	0	8	[None]	[None]	[None]	hex
2	MinSize	0	16	16	16	0	0	0	[None]	[None]	[None]	decimal
3	MaxSize	4	16	0	16	0	0	0	[None]	[None]	[None]	decimal

Fig. 9A

MinMaxSize Lookup Structure					
Protocol	**Next Index**	**Minimum**	**Maximum**	**Mask**	**Translation**
Figure 6	1	0x02	0x02	ALL	"MinMaxSize"

Fig. 9B

Filter Channel Control Structure		
System Filter Status	**Number of Filters**	**Pointer to Filter Channels**
FILTER_FRAME	1	Figure 10a

Fig. 10

Filter Channels			
Index	**NextCriteriaIndex**	**TotalCriteria**	**Criteria Pointer**	**Channel Name**
0	0	3	Figure 10b	GP Type or Station

Fig. 10A

Filter Criteria				
Index	**Channel Ptr**	**Lookup Pointer**	**Protocol Pointer**	**Field Pointer**
0	Figure 10a, Index 0	Figure 10c	Figure 4	Figure 4a, Index 0
1	Figure 10a, Index 0	Figure 10d	Figure 4	Figure 4a, Index 1
2	Figure 10a, Index 0	Figure 10e	Figure 5	Figure 5a, Index 3

Fig. 10B

Index 0 Filter Condition Lookup Structure

Return Value	NextIndex	Minimum	Maximum	Mask	Translation
FILTER_FRAME	2	0x000000	0x08FFFE	ALL	""
FILTER_FRAME	1	0x08FFFF	0x08FFFF	ALL	"Vendor XXX"
FILTER_FRAME	2	0x090000	0xFFFFFF	ALL	""

Fig. 10C

Index 1 Filter Condition Lookup Structure

Return Value	NextIndex	Minimum	Maximum	Mask	Translation
FILTER_FRAME	2	0x000000	0x334454	ALL	""
PASS_FRAME	3	0x334455	0x334455	ALL	"334455"
FILTER_FRAME	2	0x334456	0xFFFFFF	ALL	""

Fig. 10D

Index 2 Filter Condition Lookup Structure

Return Value	NextIndex	Minimum	Maximum	Mask	Translation
FILTER_FRAME	3	0x00	0x00	ALL	""
PASS_FRAME	3	0x01	0x02	ALL	"GP1 or GP2"
FILTER_FRAME	3	0x03	0xFF	ALL	""

Fig. 10E

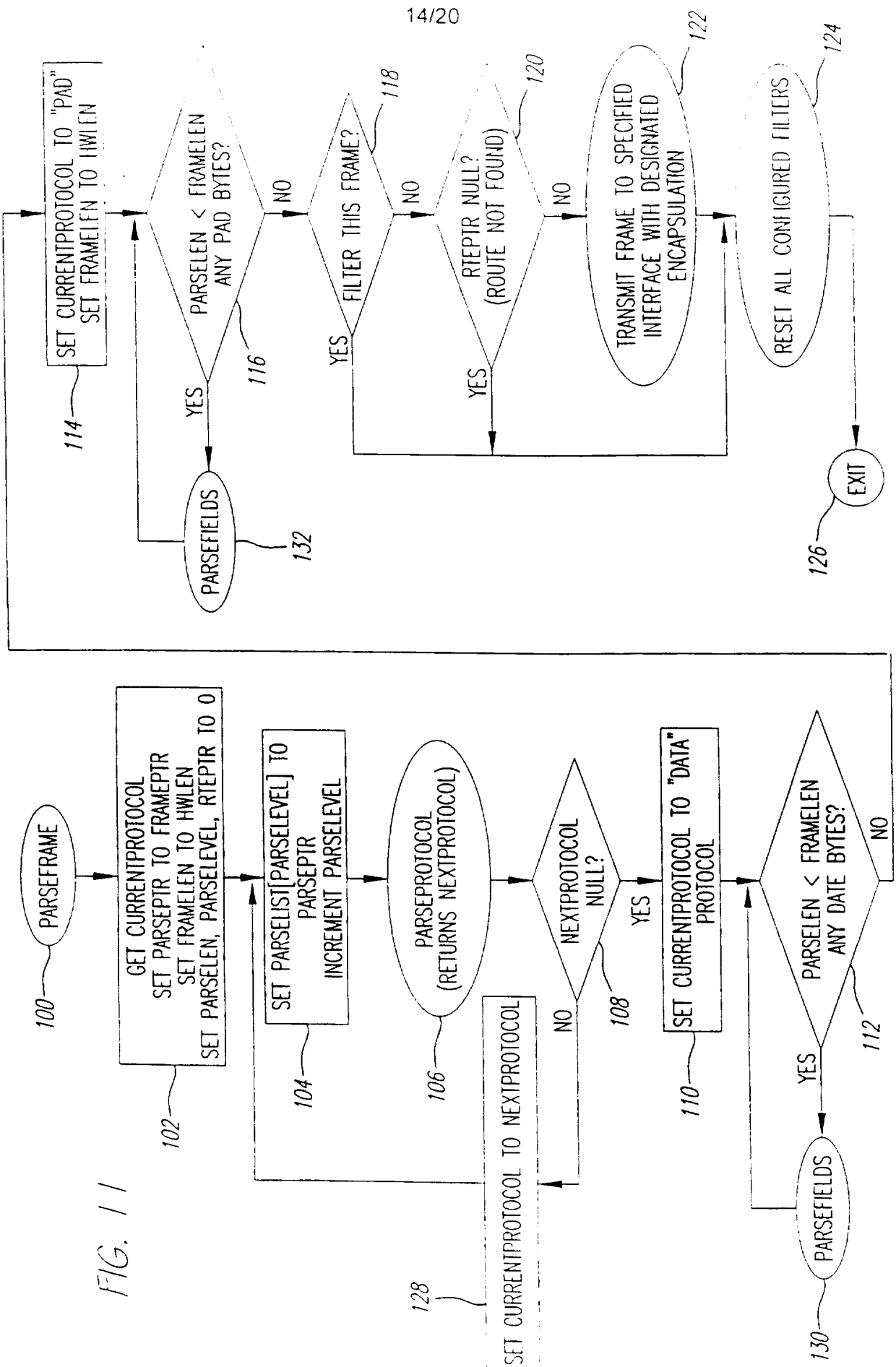


FIG. 11

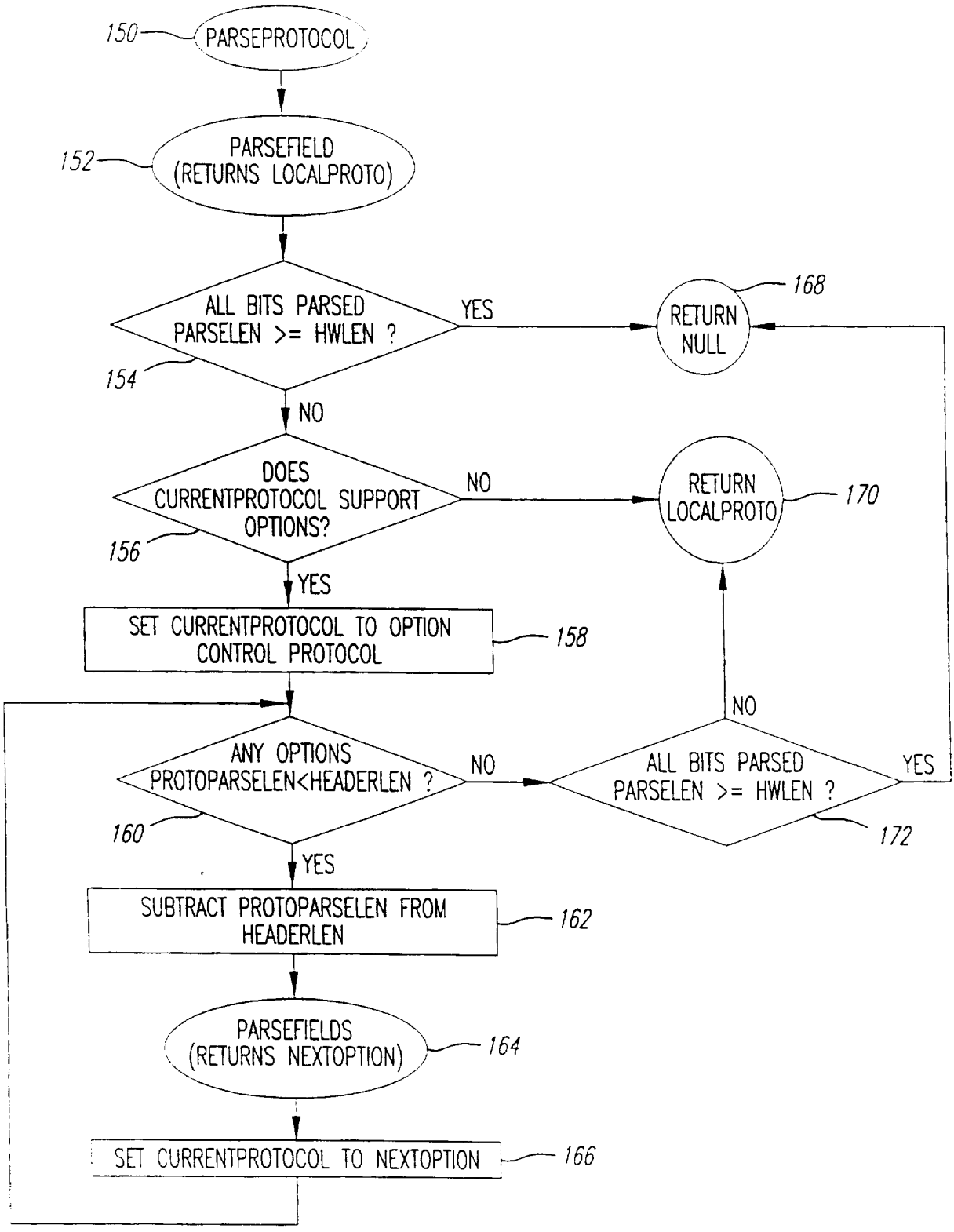


FIG. 12

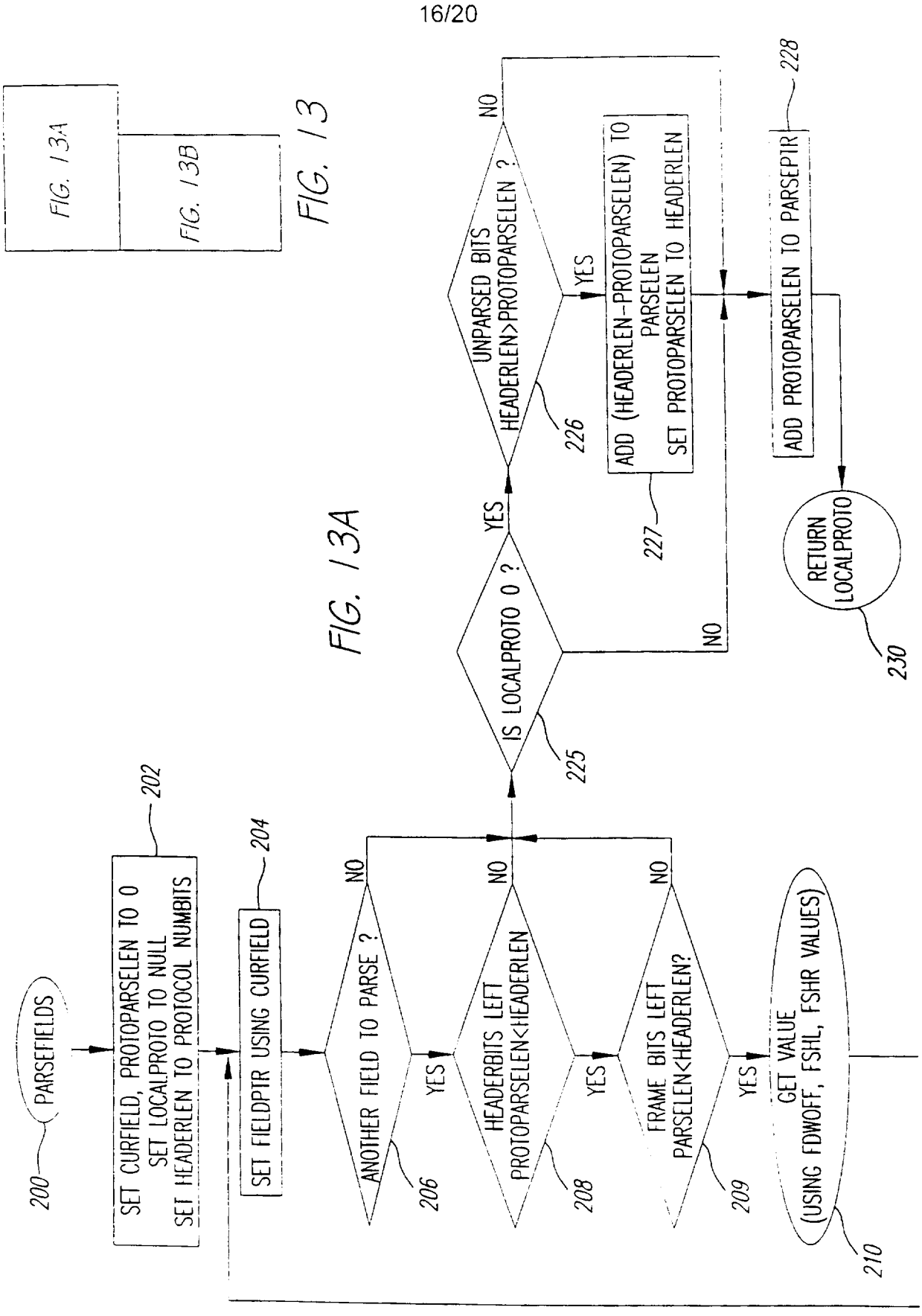
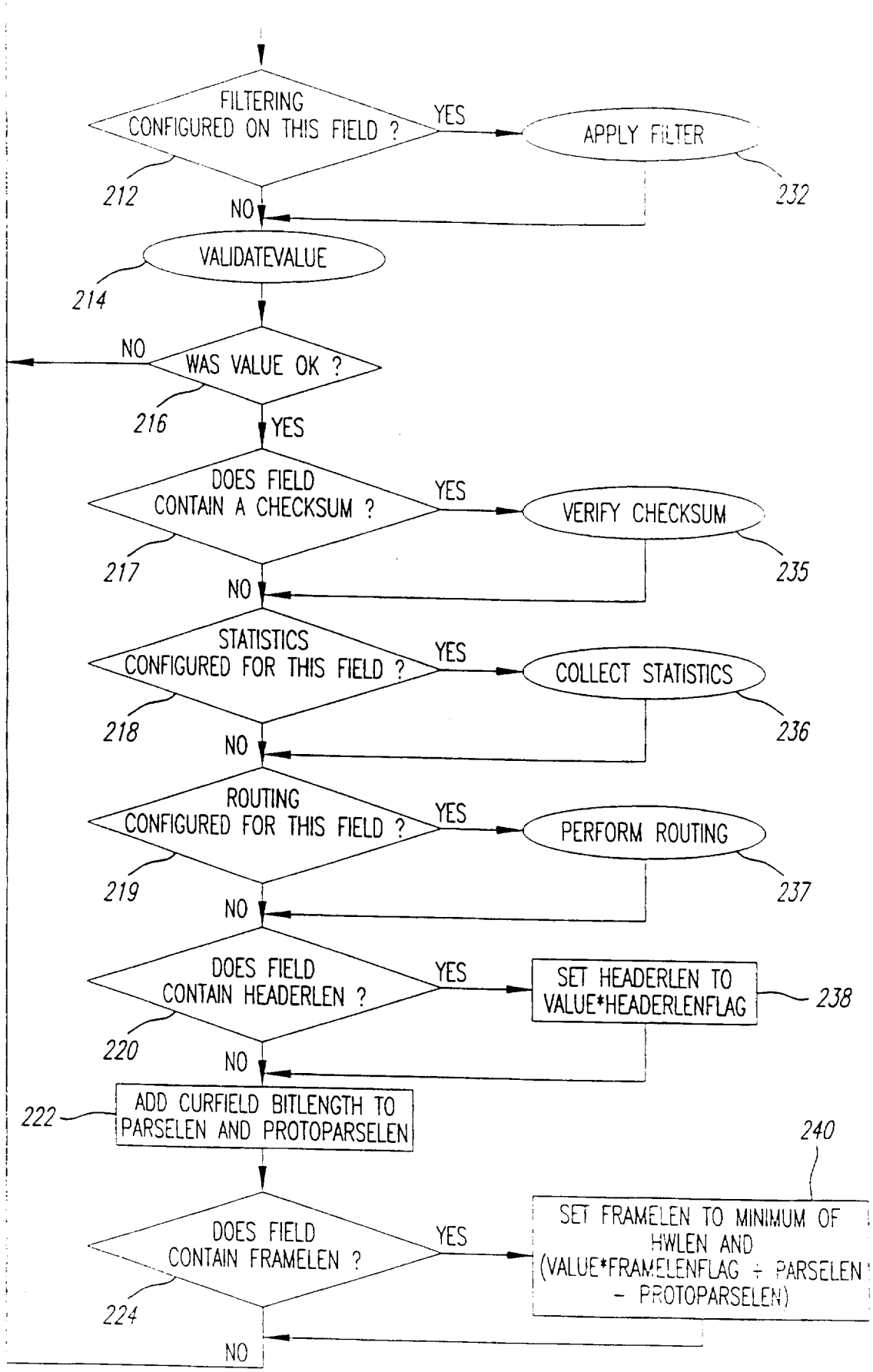


FIG. 13B


FILTERING CONFIGURED ON THIS FIELD ?
YES
APPLY FILTER
212
232
NO
VALIDATEVALUE
214
NO
WAS VALUE OK ?
216
YES
DOES FIELD CONTAIN A CHECKSUM ?
YES
VERIFY CHECKSUM
217
235
NO
STATISTICS CONFIGURED FOR THIS FIELD ?
YES
COLLECT STATISTICS
218
236
NO
ROUTING CONFIGURED FOR THIS FIELD ?
YES
PERFORM ROUTING
219
237
NO
DOES FIELD CONTAIN HEADERLEN ?
YES
SET HEADERLEN TO VALUE*HEADERLENFLAG
238
220
NO
222
ADD CURFIELD BITLENGTH TO PARSELEN AND PROTOPARSELEN
240
DOES FIELD CONTAIN FRAMELEN ?
YES
SET FRAMELEN TO MINIMUM OF HWLEN AND (VALUE*FRAMELENFLAG + PARSELEN - PROTOPARSELEN)
224
NO

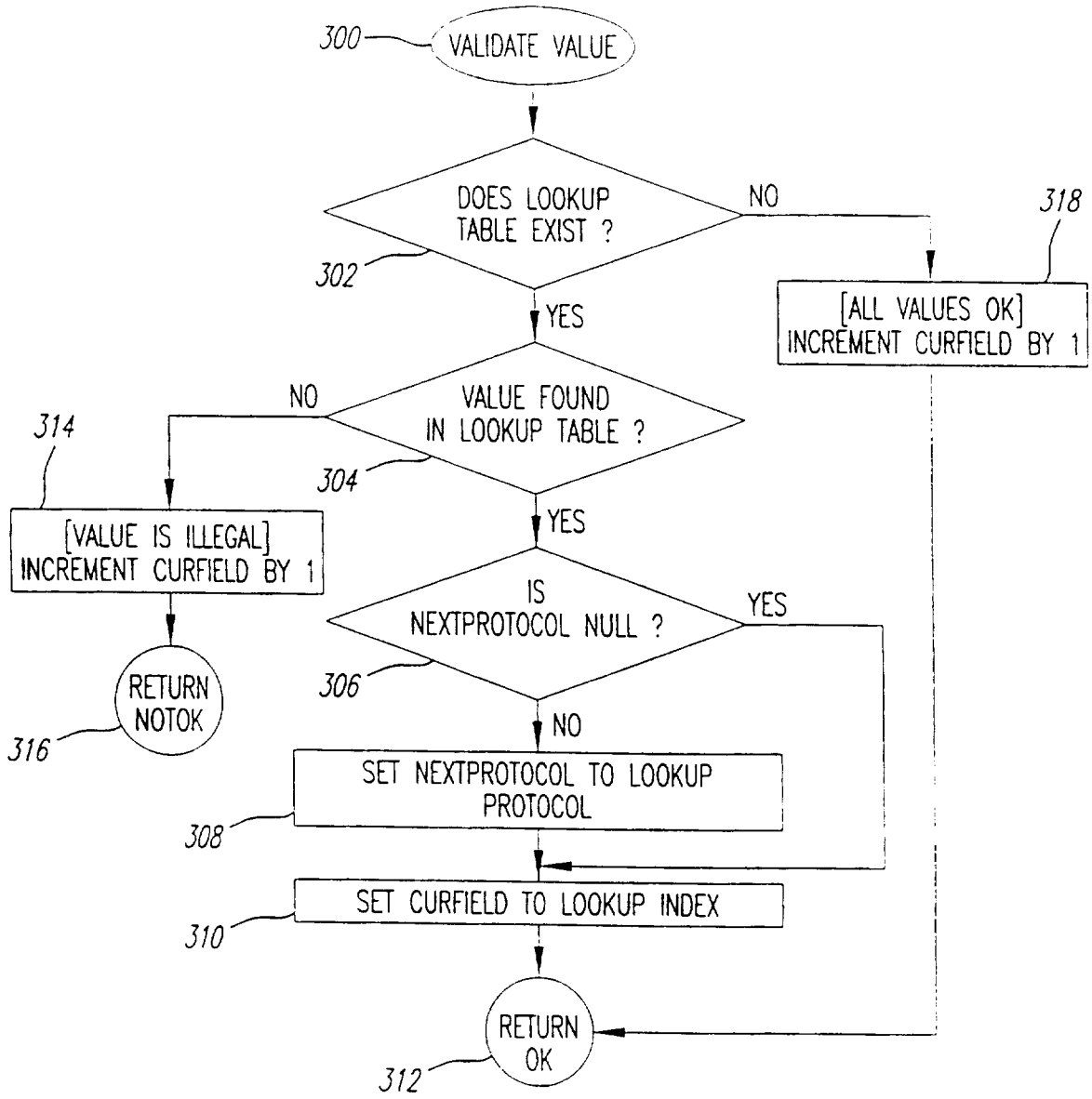


FIG. 14

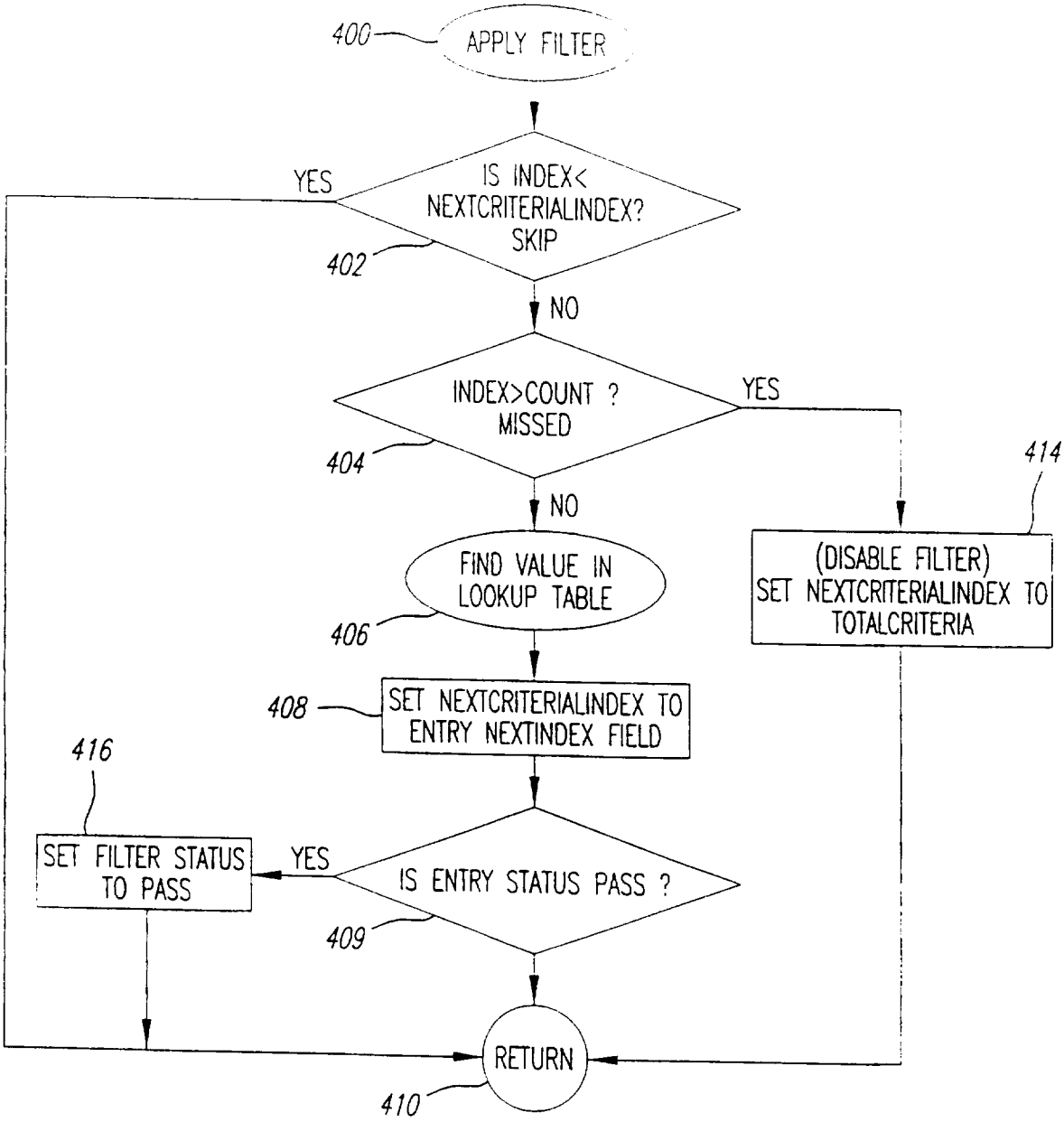


FIG. 15

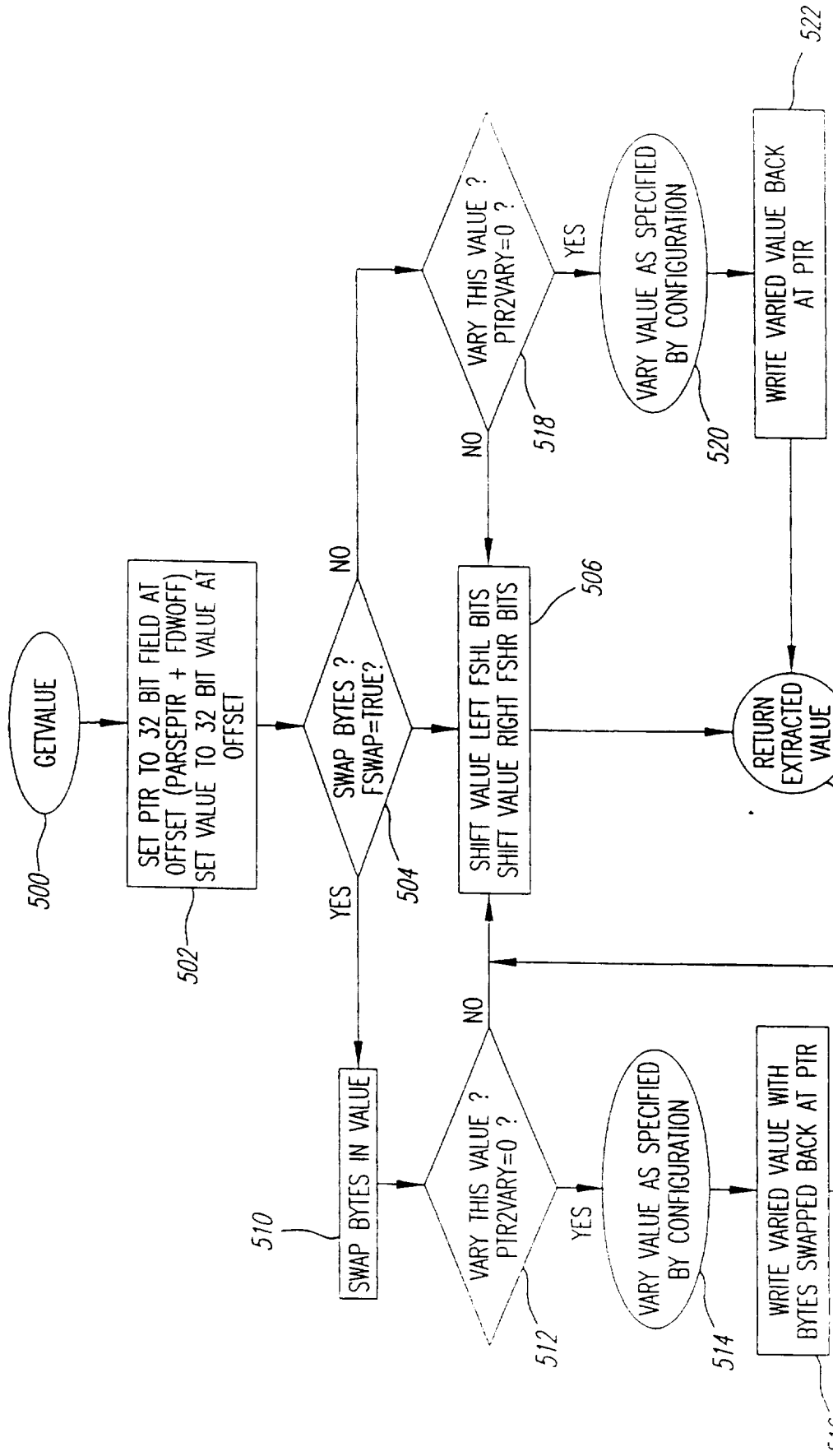


FIG. 16

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/20779

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04L 12/00
US CL :364/514C

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : Please See Extra Sheet.

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,062,055 (CHINNASWAMY ET AL) 29 October 1991, col. 45, lines 40-42, col. 6, lines 9-10 & 17, col. 46, lines 29-33, col. 6, lines 31-32, col. 5, line 50, col. 45, lines 40-42, col. 5, lines 47-48, fig. 2, item 250, fig. 2, item 210, col. 5, 39-41, col. 6, lines 24-27.	1-8, 12, 13, 15-18
Y	US, A, 5,442,639 (CROWDER ET AL) 15 August 1995, col. 7, lines 34-37, col. 4, lines 31-34, col. 9, lines 44-46, col. 9, lines 46-47, col. 10, lines 31-34, col. 10, lines 10-11.	1-3, 5, 7, 12, 13, 17, 18.

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 23 APRIL 1997	Date of mailing of the international search report 14 MAY 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer B. Hardy for THOMAS PEESO Telephone No. (703) 305-9784

Form PCT/ISA/210 (second sheet)(July 1992)★

B. FIELDS SEARCHED
Minimum documentation searched
Classification System: U.S.

364/514C, 514R, 551.01; 340/825.06; 371/35, 48, 53, 67.1, 68.2, 20.1, 3; 395/182.02, 182.19, 183.13, 183.15, 183.22, 185.01