A Basic FidoNet(tm) Technical Standard

Draft FSC001-4 - August 18, 1986

Randy Bush, Pacific Systems Group

A. Introduction

FidoNet has grown beyond most people's fantasies, and new FidoNet implementations are appearing regularly. Unfortunately, the scattered nature of the documentation and absence of clear testing procedures have made implementation difficult. The International FidoNet Association (IFNA), in its desire to promote and encourage FidoNet implementations, suggested a project to create a technical standard for FidoNet. This is the first draft document from that effort.

This document defines the data structures and communication protocols which a FidoNet implementation must provide. The implementor of FidoNet compatible systems is the intended audience of this document.

The layered metaphor of the ISO Open Systems Interface reference model has been used to view FidoNet from a standard perspective. As with most prospective ISO/OSI descriptions, FidoNet does not always make this easy.

The content of this document was gleaned from the references given at the end. This is not a a proposal, nor is it yet a standard; it is a plea for correction, more information, and other reference documents.

Please direct technical comments and errata to

Randy Bush	FidoNet	: 122/6
Pacific Systems Group	TeleMail	: RBush/PSG
601 South 12th Court	Usenet	: ..!vaxine!spark!122!6!rb
Coos Bay, Oregon US-97420	Compu$erve	: 70265,757

and administrative enquiries to

Ken Kaplan	FidoNet	: 1/0

International FidoNet Association
PO Box 41143
St. Louis, Missouri US-63141

1. Basic Requirements for a FidoNet Implementation

Compatibility is a set of abilities which, when taken as a whole, make it safe to list a net or node in the IFNA nodelist. In other words, if another node should attempt contact, does it have a reasonable chance of successful communication? This is a social obligation, as the calling system pays money for the attempt. Conversely, an implementation should be able to successfully contact other systems, as life is not a one-way street.

A FidoNet implementation must be able to call other nodes and transfer messages and files in both directions. This includes pickup and poll.

A FidoNet implementation must be able to accept calls from other nodes and transfer messages and files in both directions. This includes pickup.

A FidoNet implementation must be able to receive and process the IFNA format nodelist, and transfer nodelists to other nodes. A companion document, FSC002, defines the IFNA format nodelist and how to interpret and process it.

A FidoNet impemantation must route messages which do not have files attached through net hosts as shown in the IFNA nodelist.

2. Levels of Compliance

This documents represents the most basic FidoNet implementation. A future document will define well tested extensions which are optional but provide sufficient additional function that implementors should seriously consider them. SEAdog(tm), from System Enhancement Associates, is an excellent example of such an extended FidoNet implementation.

3. The ISO/OSI Reference Model (cribbed from "Protocol Verification via Executable Logic Specifications", D. P. Sidhu, in Rudin & West)

In the ISO/OSI model, a distributed system consists of entities that communicate with each other according to a set of rules called a protocol. The model is layered, and there are entities associated with each layer of the model which provide services to higher layers by exchanging information with their peer entities using the services of lower layers. The only actual communication between two systems is at the lowest level.

Several techniques have been used in the specification of such protocols. A common ingredient in all techniques is the notion of the extended finite state automata or machine. Extensions include the addition of state variables for the storing of state information about the protocol. The state of an automation can change as a result of one of the following events:

- Request from an upper network layer for service
- Response to the upper layer
- Request to the lower network layer to perform a service
- Response from the lower layer
- Interaction with the system and environment in which the protocol is implemented (e.g. timeouts, host operating system aborts, ...)

A protocol specification, in a large part, consists of specifying state changes in automata which model protocol entities and in describing the data which they exchange.

4. Data Description

A language specific notation was avoided. Please help stamp out environmental dependencies. Only you can prevent PClone market dominance. Don't panic, there are rectangular record layouts too.

```
(* non-terminals *)
UpperCaseName - to be defined further on

(* literals *)
"ABC"          - ASCII character string, no termination implied
nnH            - byte in hexadecimal

(* terminals *)
someName       - 16-bit integer, low order byte first (8080 style)
someName[n]    - field of n bytes
someName[.n]   - field of n bits
someName(n)    - Null terminated string allocated n chars (incl Null)
someName{max}  - Null terminated string of up to max chars (incl Null)

(* punctuation *)
a b            - one 'a' followed by one 'b'
( a | b )      - either 'a' or 'b', but not both
{ a }          - zero or more 'a's
[ b ]          - zero or one 'b'
(* comment *)  - ignored

(* predeclared constant *)
Null           = 00H
```

5. Finite State Machine Notation

State #	State Name	Predicate(s)	Action(s)	Next St
fnn*				

State # - Number of this state (e.g. R13).
- f - FSM initial (Window, Sender, Receiver, ...)
- nn - state number
- * - state which represents a lower level protocol which is represented by yet another automation.

State Name - Descriptive name of this state.

Predicate(s) - Conditions which terminate the state. If predicates are non-exclusive, consider them ordered.

Action(s) - Action(s) corresponding to predicate(s)

Next State - Subsequent state corresponding to predicate(s)

Ideally, there should be a supporting section for each state which should give a prose discription of the state, its predicates, actions, etc. So much for ideals.

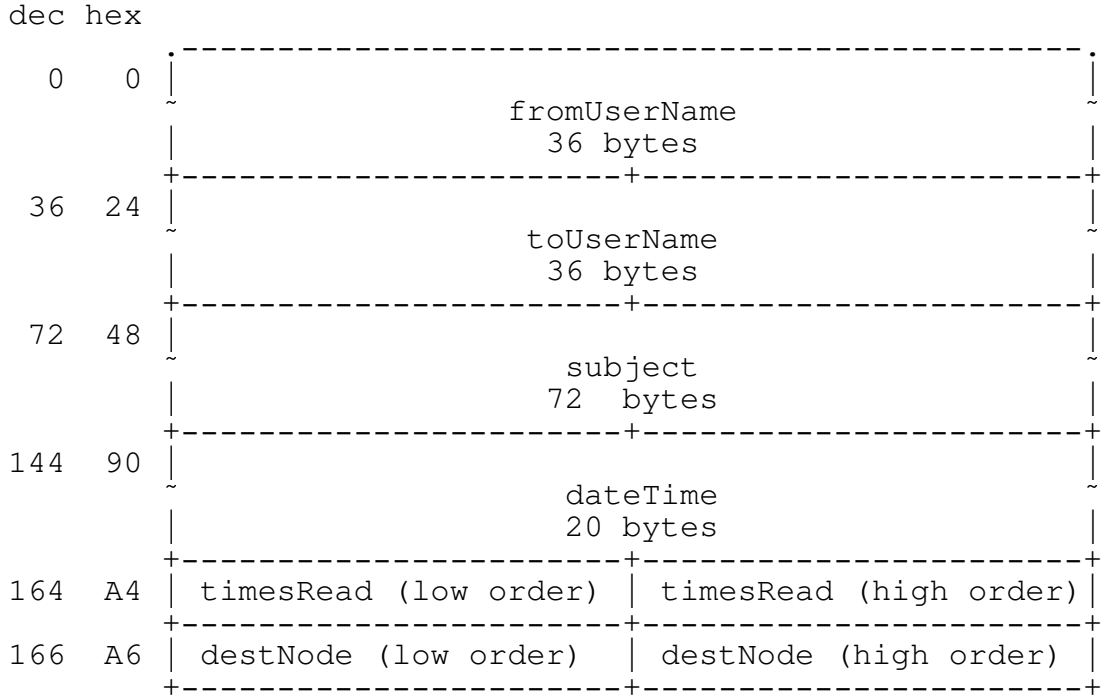
B. Application Layer : the System from the User's View

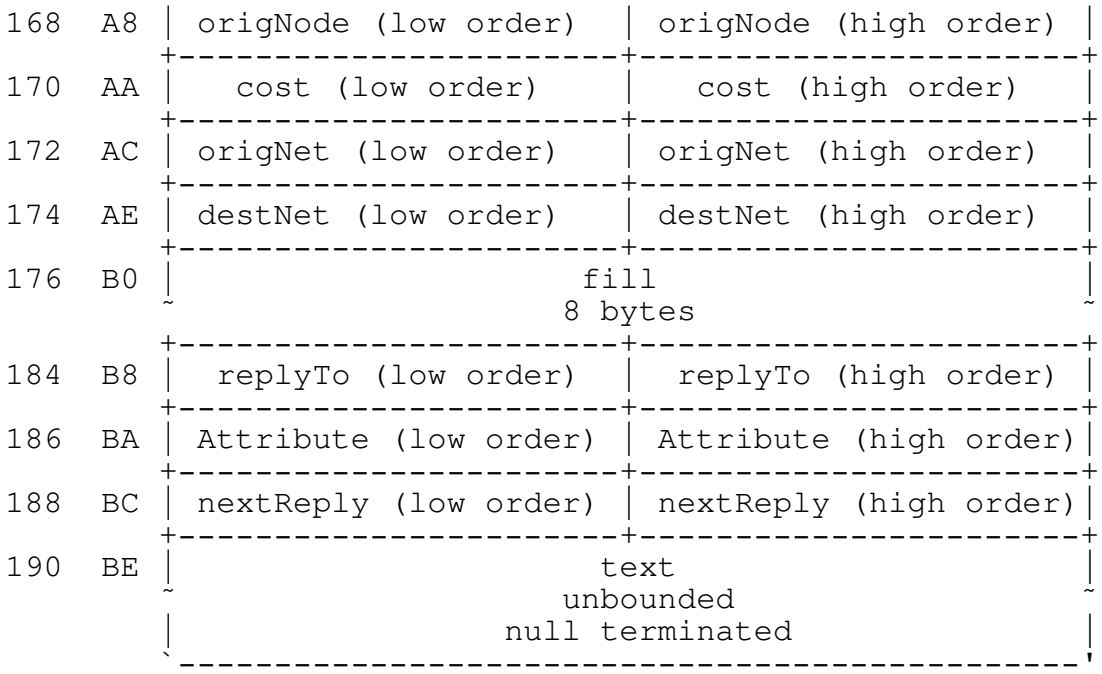
The application layer is outside the domain of a FidoNet standard, as it is the layer that the user's application sees as opposed to what FidoNet sees. In recent months, there has been sufficient confusion and discussion about the format of data at this level to warrant the description of the data structure, the message as it is stored by Fido and SEAdog.

Perfectly valid FidoNet systems may be implemented whose stored messages differ greatly from this format.

1. Application Layer Data Definition : a Stored Message

Stored Message






```
Message    = fromUserName(36)  (* Null terminated *)
             toUserName(36)    (* Null terminated *)
             subject(36)       (* see FileList below *)
             DateTime          (* message body was last edited *)
             timesRead
             destNode          (* of message *)
             origNode          (* of message *)
             cost              (* in lowest unit of originator's
                                  currency *)
             origNet           (* of message *)
             destNet           (* of message *)
             fill[8]
             replyTo           (* msg to which this replies *)
             AttributeWord
             nextReply         (* msg which replies to this *)
             text(unbounded)   (* Null terminated *)

DateTime   = (* a character string 20 characters long *)
                               (* 01 Jan 86  02:34:56 *)
             DayOfMonth " " Month " " Year " "
             " " HH ":" MM ":" SS
             Null

DayOfMonth = "01" | "02" | "03" | ... | "31"   (* Fido 0 fills *)
Month      = "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" |
             "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"
Year       = "01" | "02" | .. | "85" | "86" | ... | "99" | "00"
HH         = "00" | .. | "23"
MM         = "00" | .. | "59"
SS         = "00" | .. | "59"

AttributeWord   bit       meaning
                ---       -------------------
                  0  +    Private
```

```
 1  + s  Crash
 2       Recd
 3       Sent
 4  +    FileAttached
 5       InTransit
 6       Orphan
 7       KillSent
 8       Local
 9    s  HoldForPickup
10  +    unused
11    s  FileRequest
12  + s  ReturnReceiptRequest
13  + s  IsReturnReceipt
14  + s  AuditRequest
15    s  FileUpdateReq

      s - this bit is supported by SEAdog only
     + - this bit is not zeroed before packeting
```

Bits numbers ascend with arithmetic significance of bit position.

The message text is unbounded and null terminated (note exceptions below). <cr> (0DH) marks the end of a paragraph. <lf>s (0AH) should be ignored. Systems which display message text should wrap long lines to suit their application.

File Specifications

If one or more of FileAttached, FileRequest, or FileUpdateReq are asserted in an AttributeWord, the subject{72} field is interpreted as a list of file specifications which may include wildcards and other system-dependent data. This list is of the form

```
FileList = [ FileSpec { Sep FileSpec } ] Null

FileSpec = (* implementation dependent file specification.  may
              not contain Null or any of the characters in Sep. *)

Sep      = ( " " | "," )  { " " }
```

There are deviations from and additions to these specifications

1 - Fido does not necessarily terminate the message text with a Null, but uses an empty line (0DH 0AH 0DH 0AH)

2 - Some programs embed 'soft' carriage returns (8DH) where they fold long lines. Soft <cr>s should be politely ignored. Similarly for any <lf>s after them.

3 - SEAdog zeros the message cost field when building a message.

4 - SEAdog uses a different format for dates, e.g.

```
DateTime   = (* a character string 20 characters long *)
                 (* SEAdog format Mon  1 Jan 86 02:34 *)
             DayOfWk " " DayOfMo " " Month " " Year "
             " HH ":" MM Null

DayOfWeek  = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"
DayOfMon   = " 1" | " 2" | " 3" | ... | "31"  (* blank fill *)
```

2. Application Layer Protocol : Schedules and Events

At the application level, FidoNet imposes few protocol requirements. An implementation must automatically enter windows, or time slots, when it originates and receives node-to-node FidoNet connections. A window is described by the presentation layer.

Routing of messages will usually be different and customizable for each scheduled window.

The ability to send to and receive from any IFNA listed node during the National Mail Hour is considered mandatory.

Current implementations assemble all data for outbound connections at the start of a window, and disassemble inbound data at the end of a window. Due to performance considerations on small machines, this is considered a valid optimization. Observe that it somewhat inhibits dynamic routing.

C. Presentation Layer : the User from the System's View

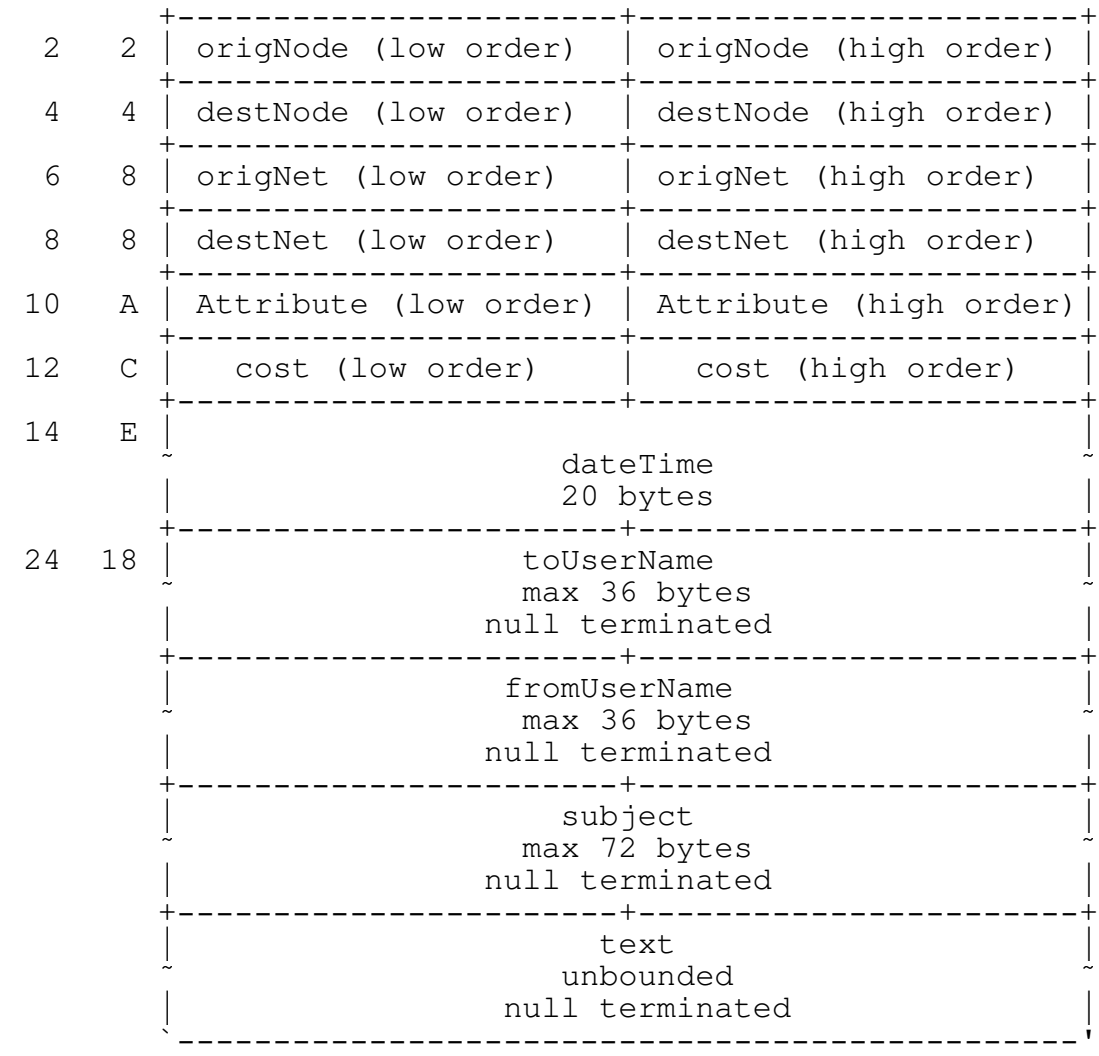
1. Presentation Layer Data Definition : the Packed Message

To conserve space and eliminate fields which would be meaningless if sent (e.g. timesRead), messages are packed for transmission. As this is a data structure which is actually transferred, its definition is critical to FidoNet. A packed message has a number of fixed length fields followed by four null terminated strings.

While most of the string fields in a stored message are fixed length, to conserve space strings are variable length when in a packet. All variable length strings are all Null terminated, including especially the message text.

Packed Message

Offset dec	Offset hex				
0	0	0	2	0	0




```
PakdMessage  = 02H 00H           (* message type, old type-1 is
                                    obsolete *)
               origNode          (* of message *)
               destNode          (* of message *)
               origNet           (* of message *)
               destNet           (* of message *)
               AttributeWord
               cost              (* in lowest unit of originator's
                                    currency *)
               DateTime          (* message body was last edited *)
               toUserName{36}    (* Null terminated *)
               fromUserName{36}  (* Null terminated *)
               subject{72}       (* Null terminated *)
               text{unbounded}   (* Null terminated *)
```

2. Presentation Layer Protocol : a Mail Window

State #	State Name	Predicate(s)	Action(s)	Next St
W0	WindTop	1 end of window reached 2 time remains in window	reset modem to not answr ensure modem can answer	exit W1
W1	WindIdle	1 incoming call 2 receive-only mode 3 send-only mode 4 60-180 secs & no call		W2 W1 W3 W3
W2*	WindRecv		(receive call R0)	W3
W3	WindCall	1 select outgoing call 2 no outgoing calls	increment try count	W4 W0
W4*	WindSend		(make call S0)	W5
W5	WindMark	1 call successful 2 no connect 3 call failed	remove node fr call list remove if try cnt > lim incr conn cnt, remove if con cnt > lim	W0 W0 W0

D. Session Layer : Connecting Two FidoNet Systems

1. Session Layer Protocol : Connecting to Another FidoNet Machine

A session is a connection between two FidoNet machines. It is currently assumed to be over the DDD telephone network via modems. The calling machine starts out as the sender and the called machine as the receiver. The pickup feature is described by the sender and receiver changing roles midway through the session, after the sender has transferred the message packet and any attached files. Due to the lack of security in

the pickup protocol (danger of pickup by a fake node), a change in the protocol may be expected in the near future.

Sender

State #	State Name	Predicate(s)	Action(s)	Next St
S0	SendInit		dial modem	S1
S1	WaitCxD	1 carrier detected 2 busy, etc. 3 voice 4 carrier not detected within 60 seconds	delay 5 seconds report no connection report no carrier report no connection	S2 exit exit exit
S2	WhackCRs	1 over 30 seconds 2 ?? <cr>s received 3 <cr>s not received	report no response <cr> delay 1 sec send <cr> <sp> <cr> <sp> delay ??? secs	exit S3 S2
S3	WaitClear	1 no input for 0.5 secs 2 over 60 seconds and line not clear	send TSYNCH = AEH hang up, report garbage	S4 exit
S4*	SendMail		(XMODEM send packet XS0)	S5
S5	CheckMail	1 XMODEM successful 2 XMODEM fail or timeout	(Fido registers success) hang up, report mail bad	S6 exit
S6	SendAtt	1 more files attached 2 no more files	 send EOT	S7 S9
S7*	SendFile		(BATCH send a file BS0)	S8
S8	CheckFile	1 BATCH send successful 2 BATCH send failed	 hang up, rept files fail	S6 exit
S9	TryPickup	1 wish to pickup 2 no desire to pickup	note send ok delay 5 secs hang up, rept send ok	R2* exit

Although the above shows the sender emitting only one TSYNCH, it is recommended that a timeout of 5-20 seconds should initiate another TSYNCH. The receiver should tolerate multiple TSYNCHs.

Receiver

The receiving FSM is given an external timer, the expiration of which will cause termination with a result of 'no calls' (R0.2).

State #	State Name	Predicate(s)	Action(s)	Next St
R0	WaitCxD	1 carrier detected 2 external timer expires	delay ??? seconds report no calls	R1 exit
R1	WaitBaud	1 baud rate detected 2 no detect in ?? secs	send signon with <cr>s hang up, report no baud	R2 exit
R2	WaitTsync	1 TSYNCH received 2 60 seconds timeout	ignore input not TSYNCH hang up, report not Fido	R3 exit
R3*	RecMail		(XMODEM rec packet XR0)	R4
R4	XRecEnd	1 XMODEM successful 2 XMODEM failed	delay 1 second flush input hang up, rept mail fail	R5 exit
R5*	RecFiles		(BATCH rec files BR0)	R6
R6	ChkFiles	1 BATCH recv successful 2 BATCH recv failed	delay 2 secs hang up, report bad file	R7 exit
R7	AllowPkup	1 have pickup for sender 2 nothing to pickup	receiver becomes sender hang up, rept recv ok	S3* exit

E. Transport Layer : ?????

1. Data Definitions

2. Tranport Layer Protocol : Routing

FidoNet does not necessarily send a message directly to its destination. To reduce the number of network connections, mail to a subset of the nodelist may be routed to one node for further distribution within that subset. In addition, custom routing is possible. Routing of a message is determined in one of three ways.

- o If there are files attached, then a message must be sent directly to its destination.

- o Messages without attached files should be routed through the inbound host of the destination node's subnet as specified by an IFNA format nodelist.

- o To prevent overloading of inbound hosts, a system should provide for host routing to be disabled for a target node, or nodes.

F. Network Layer : the Network's View of the System, Routing and Packets

1. Network Layer Data Definition : the Packet Header

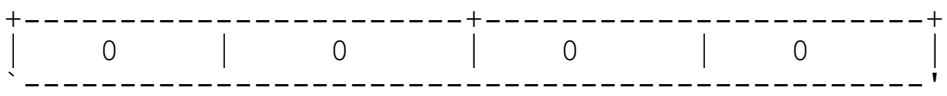
The packet contains messages in packed format to be transferred over the net during a connection. As this data structure is transferred, its definition is critical to FidoNet.

A packet may contain zero or more packed messages. A packet without messages is often generated as a poll packet.

Every packet begins with a packet header. The fields of the packet header are of fixed length.

Packet Header

Offset dec	Offset hex		
0	0	origNode (low order)	origNode (high order)
2	2	destNode (low order)	destNode (high order)
4	4	year (low order)	year (high order)
6	6	month (low order)	month (high order)
8	8	day (low order)	day (high order)
10	A	hour (low order)	hour (high order)
12	C	minute (low order)	minute (high order)
14	E	second (low order)	second (high order)
16	10	baud (low order)	baud (high order)
18	12	0	2	0	0
20	14	origNet (low order)	origNet (high order)
22	16	destNet (low order)	destNet (high order)
24	18	ProductCode	
		fill 33 bytes	
58	3A	zero or more packed messages	




```
Packet       = PacketHeader  { PakdMessage }  00H 00H

PacketHeader = origNode   (* of packet, not of messages in packet *)
               destNode   (* of packet, not of messages in packet *)
               year       (* of packet creation, e.g. 1986 *)
               month      (* of packet creation, 0-11 for Jan-Dec *)
               day        (* of packet creation, 1-31 *)
               hour       (* of packet creation, 0-23 *)
               minute     (* of packet creation, 0-59 *)
               second     (* of packet creation, 0-59 *)
               baud       (* max baud rate of orig and dest, 0=SEA *)
               PacketType (* old type-1 packets now obsolete *)
               origNet    (* of packet, not of messages in packet *)
               destNet    (* of packet, not of messages in packet *)
               ProductCode(* 0 for both Fido and SEAdog *)
               fill[33]

PacketType   = 02H 00H  (* 01H 00H was used by Fido versions before 10
                           which did not support local nets.  The
packed
                           message header was also different for those
                           versions *)

ProductCode  = (  00H      (* Fido *)
               |  00H      (* SEAdog!? *)
               |  ??H      (* Colossus *)
               |  ??H      (* ?? *)
               )
```

The remainder of the packet consists of packed messages. Each packed message begins with a message type word 0200H. A pseudo-message beginning with the word 0000H signifies the end of the packet.

2. Network Layer Data Description : a File with Attributes

The BATCH protocol uses the MODEM7 filename and TeLink/XMODEM file transfer protocols to transfer the file with attributes.

When a file is transferred via FidoNet, an attempt is made to also pass the operating system's attributes for the file such as length, modification date, etc. FidoNet does this via a special prefix block to the XMODEM file transfer using a protocol known as TeLink. As the TeLink protocol relies on a modification to the XMODEM file transfer protocol, it is documented at the data link layer level.

The MODEM7 file name is redundant if there is also a TeLink block, in which case the name may be taken from either or both.

FileName as Sent

Offset dec	Offset hex	
0	0	fileName 8 bytes left adjusted blank filled
8	8	fileExt 3 bytes left adjusted blank filled

3. Network Layer Protocol : BATCH File Finite State Machines

BATCH File Sender

State #	State Name	Predicate(s)	Action(s)	Next St
BS0*	SendFName		(MODEM7 FName send MS0)	BS1
BS1	CheckFNm	1 MODEM7 Filename ok	(TeLink send file XS0)	BS2

		2 MODEM7 Filename bad	report name send bad	exit
BS2	CheckFile	1 TeLink send ok	report file send ok	exit
		2 TeLink send bad	report file send bad	exit

BATCH File Receiver

State #	State Name	Predicate(s)	Action(s)	Next St
BR0*	RecvName		(MODEM7 FName recv MR0)	BR1
BR1	CheckFNm	1 MODEM7 no more files	report files recd ok	exit
		2 MODEM7 Filename ok	(TeLink recv file XR0)	BR2
		2 MODEM7 Filename bad	report name recv bad	exit
BR2	CheckFile	1 TeLink recv ok	report file recv ok	exit
		2 TeLink recv bad	report file recv bad	exit

G. Data Link Layer : Error-Free Data Transfer

1. Data Link Layer Data Definition : XMODEM/TeLink Blocks

XMODEM transfers are in blocks of 128 uninterpreted data bytes preceeded by a three byte header and followed by either a one byte checksum or a two byte crc remainder. XMODEM makes no provision for data streams which are not an integral number of blocks long. Therefore, the sender must pad with zeros streams whose length is not a multiple of 128 bytes, and use some other means to convey the true data length to the receiver (e.g. TeLink file info block).

The sender returns an EOT instead of a data block when no data remain.

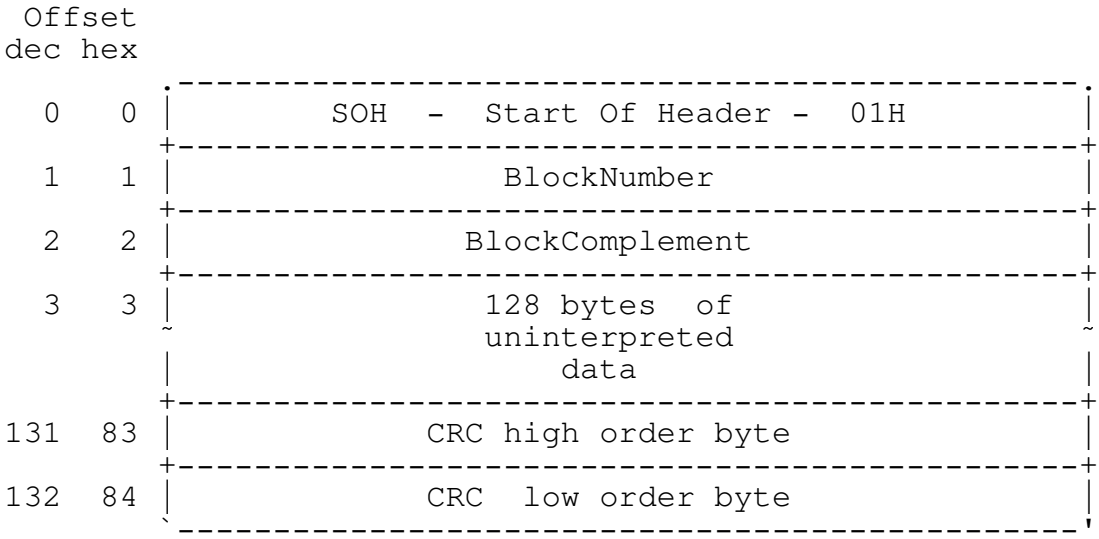
The protocol is receiver driven, the receiver polling the sender for each block. If the receiver polls for the first block using a "C" (43H) as the poll character, it would prefer to have the CRC-CCITT polynomial remainder error detection code at the end of each block as opposed to a one byte unsigned checksum. The sender will respond to the "C" poll iff it can comply. If the sender chooses checksum as opposed to CRC, it waits for the receiver to poll with NAK (15H). Should the checksum method be preferable to the receiver, it polls with NAK rather than "C".

Data blocks contain sequence numbers so the receiver can ensure it has the correct block. Block numbers are sequential unsigned eight bit integers beginning with 01H and wrapping to 00H, except that a TeLink block is given sequence number 00H.

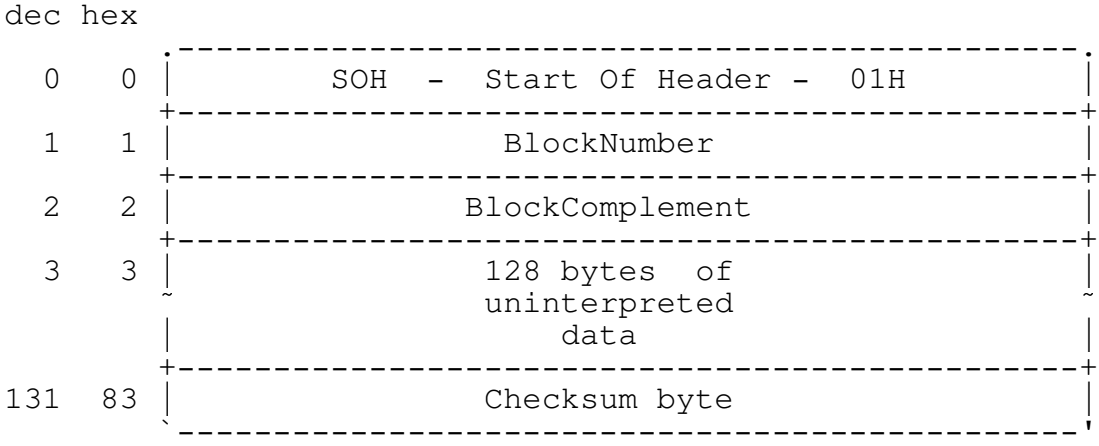
For files which are attached to the mail packet, not the mail packet itself, if the sending system is aware of the file attributes as they are known to the operating system, then the first block of the XMODEM transfer may be a special TeLink block to transfer that information. This block differs in that the first byte is a SYN character as

opposet to an SOH. Should the receiver be unwilling to handle such information, after two NAKs (or "C"s), the sender skips this special block and goes on to the data itself.

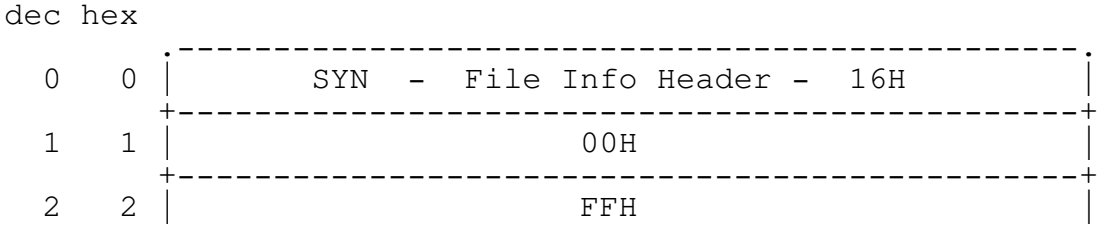
XMODEM Data Block (CRC mode)

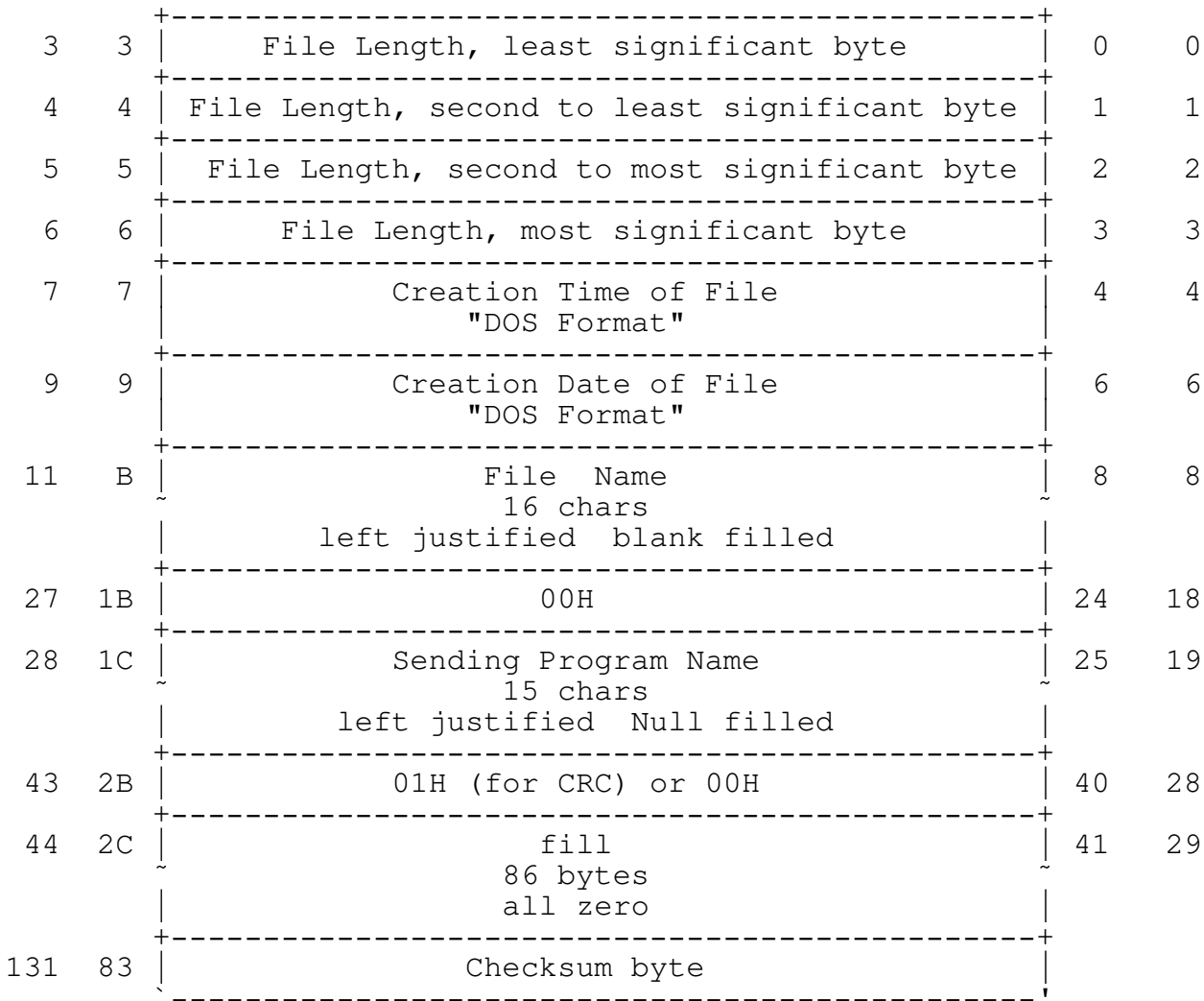


XMODEM Data Block Checksum mode)



TeLink File Descriptor Block






```
XMODEMData   = XMODEMBlock       (* block of data with header and
                                    trailer *)
               | TeLinkBlock     (* TeLink File Descriptor Block *)
               | ACK             (* acknowledge data received ok *)
               | NAK             (* negative ACK & poll 1st block *)
               | EOT             (* end of xfer, after last block *)
               | "C"             (* 43H *)

XMODEMBlock  = SOH               (* Start of Header, XMODEM Block *)
               blockNumber[1]    (* sequence, i'=mod( i+1, 256 ) *)
               blockCompl[1]     (* one's compl of BlockNumber *)
               data[128]         (* uninterpreted user data block *)
               (CRC | Checksum)  (* error detect/correction code *)

TeLinkBlock  = SYN               (* File Info Header *)
               00H               (* block no, must be first block *)
               FFH               (* one's complement of block no *)
               fileLength[4]     (* length of data in bytes *)
               CreationTime[2]   (* time file last modified or zero *)
               CreationDate[2]   (* date file last modified or zero *)
               fileName(16)      (* name of file, not vol or dir *)
```

```
                  00H                (* header version number *)
                  sendingProg(16)    (* name of program on send side *)
                  crcMode[1]         (* 01H for CRC 00H for Checksum *)
                  fill[87]           (* zeroed *)
                  (CRC | Checksum)   (* error detect/correction code *)

  ACK           = 05H                (* acknowledge data received ok *)
  NAK           = 15H                (* negative ACK & poll 1st block *)
  SOH           = 01H                (* start of header, begins block *)
  SYN           = 16H                (* start of TeLink file info blk *)
  EOT           = 04H                (* end of xfer, after last block *)

  CRC           = crc[2]             (* CCITT Cyclic Redundancy Check *)

  Checksum      = checksum[1]        (* low 8 bits of sum of data bytes
                                        using unsigned 8 bit arithmetic *)

  CreationDate  = year[.7]           (* 7 bits, years since 1980, 0-127  *)
                  month[.4]          (* 4 bits, month of year, 1-12 *)
                  day[.5]            (* 5 bits, day of month, 1-31 *)

  CreationTime  = hour[.5]           (* 5 bits, hour of day, 0-23 *)
                  minute[.6]         (* 6 bits, minute of hour, 0-60 *)
                  biSeconds[.2]      (* 6 bits, seconds/2, 0-28 *)
```


2. Data Link Layer Protocol : XMODEM/TeLink Finite State Machines

XMODEM/TeLink Sender

State #	State Name	Predicate(s)	Action(s)	Next St
XS0	WaitTeLnk	1 over 40-60 seconds	report sender timeout	exit
		2 over 2 tries	note TeLink block failed	XS1
		3 NAK or "C" received	send TeLink, incr tries	XS0
		4 ACK received	TeLink ok, set crc/cksm	XS1
XS1	WaitStart	1 over 40-60 seconds	report sender timeout	exit
		2 over 20 tries	report send failed	exit

		3 NAK received	set checksum mode	XS2
		4 "C" recd, I can crc	set crc mode	XS2
		5 "C" recd, I can't crc		XS1
XS2	SendBlock	1 more data available	send next data block as checksum or crc	XS3
		2 last block has gone	send EOT	XS4
XS3	WaitACK	1 10 retries or 1 minute	report send failed	exit
		2 ACK received		XS2
		3 NAK received	resend last block	XS3
XS4	WaitEnd	1 10 retries or 1 minute	report send failed	exit
		2 ACK received	report send successful	exit
		3 NAK received	resend EOT	XS4

XMODEM/TeLink Receiver

State #	State Name	Predicate(s)	Action(s)	Next St
XR0	RecStart	1 prefer crc mode	Send "C"	XR1
		2 want checksum mode	send NAK	XR1
XR1	WaitFirst	1 10 retries or 1 minute	report receive failure	exit
		2 > 3 retries or 30 secs	set want checksum mode	XR0
		3 EOT received	send ACK, report no file	exit
		4 TeLink block recd	send ACK, set crc/cksm	XR2
		5 data block recd	send ACK, set crc/cksm	XR2
		6 bad block or 2-10 secs	incr retry count	XR0
XR2	WaitBlock	1 10 retries or 1 minute	report receive failure	exit
		2 EOT received	send ACK, report recd ok	exit
		3 data block received	send ACK	XR2
		4 bad block or 2-10 secs	send NAK, incr retry cnt	XR2

3. Data Link Layer Protocol : MODEM7 Filename Finite State Machines

MODEM7 Filename Sender

State #	State Name	Predicate(s)	Action(s)	Next St
MS0	WaitNak	1 20 retries or 1 minute	filename send failed	exit

		2 NAK received	send ACK & 1st ch of fn	MS1
MS1	WaitChAck	1 ACK rcd, fname done	send SUB	MS2
		2 ACK rcd, fname ~done	send next ch of fname	MS1
		3 other char or 1 sec	send "u", incr retry cnt	MS0
MS2	WaitCksm	1 cksum recd and ok	send ACK, report fn ok	exit
		2 cksum recd but bad	send "u", incr retry cnt	MS0
		3 no cksum in 1 sec	send "u", incr retry cnt	MS0

MODEM7 Filename Receiver

State #	State Name	Predicate(s)	Action(s)	Next St
MR0	SendNak	1 20 tries or 1 minute	report filename failure	exit
		2	send NAK, incr try cnt	MR1
MR1	WaitAck	1 rcd ACK		MR2
		2 rcd EOT	report no files remain	exit
		3 5 secs & no ACK/EOT		MR0
MR2	WaitChar1	1 recd EOT (can happen?)	report no files remain	exit
		2 recd SUB	send checksum byte	MR3
		3 recd "u"		MR0
		4 recd char of name	send ACK	MR2
		5 no char in 1 second		MR0
MR3	WaitOkCk	1 recd ACK within 1 sec	report recd filename ok	exit
		2 recd "u" or other char		MR0

The checksum is the unsigned low oder eight bits of the sum of the character in the transferred filename including the SUB.

Will one of the more hardware-oriented comm types give me some idea of what's needed here? Can we leave it open enough to allow implementation over a non-dial net? Thanks.

I. The Node List
  1. Node List Format
  2. Node List Processing
  3. Node Diff Format
  4. Node Diff Processing

J. Tests to Validate an Implementation
  - o It would be ideal if validation were automatic
  - o Implementors could use in last stages of development
  - o Program(s)? Test scripts? Remote Fidos? Bad line conditions?

K. Example programs
  - o Bob Pritchett has offered
  - o Free to implementors et al
  - o Likely in C

L. Acknowledgements

Ben Baker, Thom Henderson, Tom Jennings, and Gee Wong suggested, informed, reviewed, and encouraged. Thom and Tom allowed me to look at actual code.

My employer, Pacific Systems Group was kind enough to donate my time to research and to write this document.

M. Bibliography

Documentation for the protocols and data formats are scattered. Some are unattributed, some even untitled. Filenames indicate that softcopy is available from Fido 122/6 and likely many others.

Anonymous, changes to MODEM to impmlement CRC option XMDM-CRC.TXT

Christensen, Ward, "MODEM Protocol Overview" of 1 January 82 XMODEM.TXT

Henderson, Thom, "SEAdog Electronic Mail System Version 3" of April 86

International Standards Organization, "Data Processing - Open Systems Interconnection - Basic Reference Model" ISO/DIS 7498 April 82

Jennings, Tom, "FidoNet Electronic Mail Protocol" 8 February 85 FIDOMAIL.DOC

Jennings, Tom, "Fido's Internal Structures" of 13 September 85 STRUCT.TXT aka STRUCT.APX

Jennings, Tom, "Extending XMODEM/MODEM File Transfer Protocol to support DOS" 20 September 83 FILEXFER.DOC

Jordan, Larry, "XMODEM File Transfer Protocol" XMDM-LJ.TXT

Rudin, H and West, C, "Protocol Specification, Testing, and Verification, III" Proceedings of the IFIP WG 6.1 Third International Workshop on Protocol Specification, Testing, and Verification, Rueschlikon Switzerland 31 May - 2 June 1983.

Tanenbaum, Andrew, "Computer Networks" Prentice Hall 1981

Messages generated by Fido 11w, SEAdog 3.8, and ARCmail 0.37

→