# XMODEM/YMODEM PROTOCOL REFERENCE
A compendium of documents describing the

XMODEM and YMODEM

File Transfer Protocols

This document was formatted 10-2-88.

Edited by Chuck Forsberg

Questions to Chuck Forsberg

Omen Technology Inc
The High Reliability Software
17505-V Sauvie Island Road
Portland Oregon 97231
VOICE: 503-621-3406 :VOICE
TeleGodzilla BBS: 503-621-3746 Speed 19200(Telebit PEP),2400,1200,300
CompuServe: 70007,2304
GEnie: CAF
UUCP: ...!tektronix!reed!omen!caf

# 1. TOWER OF BABEL

A "YMODEM Tower of Babel" has descended on the microcomputing community bringing with it confusion, frustration, bloated phone bills, and wasted man hours. Sadly, I (Chuck Forsberg) am partly to blame for this mess.

As author of the early 1980s batch and 1k XMODEM extensions, I assumed readers of earlier versions of this document would implement as much of the YMODEM protocol as their programming skills and computing environments would permit. This proved a rather naive assumption as programmers motivated by competitive pressure implemented as little of YMODEM as possible. Some have taken whatever parts of YMODEM that appealed to them, applied them to MODEM7 Batch, Telink, XMODEM or whatever, and called the result YMODEM.

Jeff Garbers (Crosstalk package development director) said it all: "With protocols in the public domain, anyone who wants to dink around with them can go ahead." [1]

Documents containing altered examples derived from YMODEM.DOC have added to the confusion. In one instance, some self styled rewriter of history altered the heading in YMODEM.DOC's Figure 1 from "1024 byte Packets" to "YMODEM/CRC File Transfer Protocol". None of the XMODEM and YMODEM examples shown in that document were correct.

To put an end to this confusion, we must make "perfectly clear" what YMODEM stands for, as Ward Christensen defined it in his 1985 coining of the term.

To the majority of you who read, understood, and respected Ward's definition of YMODEM, I apologize for the inconvenience.

## 1.1 Definitions

| | |
|---|---|
| ARC | ARC is a program that compresses one or more files into an archive and extracts files from such archives. |
| XMODEM | refers to the file transfer etiquette introduced by Ward Christensen's 1977 MODEM.ASM program. The name XMODEM comes from Keith Petersen's XMODEM.ASM program, an adaptation of MODEM.ASM for Remote CP/M (RCPM) systems. It's also called the MODEM or MODEM2 protocol. Some who are unaware of MODEM7's unusual batch file mode call it MODEM7. Other aliases include "CP/M Users' Group" and "TERM II FTP 3". The name XMODEM caught on partly because it is distinctive and partly because of media interest in |

1. Page C/12, PC-WEEK July 12, 1987

```
        bulletin board and RCPM systems where it was accessed with an
        "XMODEM" command.  This protocol is supported by every serious
        communications program because of its universality, simplicity,
        and reasonable performance.

XMODEM/CRC replaces XMODEM's 1 byte checksum with a two byte Cyclical
        Redundancy Check (CRC-16), giving modern error detection
        protection.

XMODEM-1k Refers to the XMODEM/CRC protocol with 1024 byte data blocks.

YMODEM  Refers to the XMODEM/CRC (optional 1k blocks) protocol with batch
        transmission as described below.  In a nutshell, YMODEM means
        BATCH.

YMODEM-g Refers to the streaming YMODEM variation described below.

True YMODEM(TM) In an attempt to sort out the YMODEM Tower of Babel, Omen
        Technology has trademarked the term True YMODEM(TM) to represent
        the complete YMODEM protocol described in this document, including
        pathname, length, and modification date transmitted in block 0.
        Please contact Omen Technology about certifying programs for True
        YMODEM(TM) compliance.

ZMODEM  uses familiar XMODEM/CRC and YMODEM technology in a new protocol
        that provides reliability, throughput, file management, and user
        amenities appropriate to contemporary data communications.

ZOO     Like ARC, ZOO is a program that compresses one or more files into
        a "zoo archive".  ZOO supports many different operating systems
        including Unix and VMS.
```

## 2. YMODEM MINIMUM REQUIREMENTS

All programs claiming to support YMODEM must meet the following minimum requirements:

+ The sending program shall send the pathname (file name) in block 0.

+ The pathname shall be a null terminated ASCII string as described below.

  For those who are too lazy to read the entire document:

  + Unless specifically requested, only the file name portion is sent.

  + No drive letter is sent.

  + Systems that do not distinguish between upper and lower case letters in filenames shall send the pathname in lower case only.

+ The receiving program shall use this pathname for the received file name, unless explicitly overridden.

+ When the receiving program receives this block and successfully opened the output file, it shall acknowledge this block with an ACK character and then proceed with a normal XMODEM file transfer beginning with a "C" or NAK tranmsitted by the receiver.

+ The sending program shall use CRC-16 in response to a "C" pathname nak, otherwise use 8 bit checksum.

+ The receiving program must accept any mixture of 128 and 1024 byte blocks within each file it receives. Sending programs may arbitrarily switch between 1024 and 128 byte blocks.

+ The sending program must not change the length of an unacknowledged block.

+ At the end of each file, the sending program shall send EOT up to ten times until it receives an ACK character. (This is part of the XMODEM spec.)

+ The end of a transfer session shall be signified by a null (empty) pathname, this pathname block shall be acknowledged the same as other pathname blocks.

Programs not meeting all of these requirements are not YMODEM compatible, and shall not be described as supporting YMODEM.

Meeting these MINIMUM requirements does not guarantee reliable file transfers under stress.  Particular attention is called to XMODEM's single character supervisory messages that are easily corrupted by transmission errors.

# Chapter 2

## 3. WHY YMODEM?

Since its development half a decade ago, the Ward Christensen modem protocol has enabled a wide variety of computer systems to interchange data. There is hardly a communications program that doesn't at least claim to support this protocol.

Advances in computing, modems and networking have revealed a number of weaknesses in the original protocol:

+ The short block length caused throughput to suffer when used with timesharing systems, packet switched networks, satellite circuits, and buffered (error correcting) modems.

+ The 8 bit arithmetic checksum and other aspects allowed line impairments to interfere with dependable, accurate transfers.

+ Only one file could be sent per command. The file name had to be given twice, first to the sending program and then again to the receiving program.

+ The transmitted file could accumulate as many as 127 extraneous bytes.

+ The modification date of the file was lost.

A number of other protocols have been developed over the years, but none have displaced XMODEM to date:

+ Lack of public domain documentation and example programs have kept proprietary protocols such as Blast, Relay, and others tightly bound to the fortunes of their suppliers.

+ Complexity discourages the widespread application of BISYNC, SDLC, HDLC, X.25, and X.PC protocols.

+ Performance compromises and complexity have limited the popularity of the Kermit protocol, which was developed to allow file transfers in environments hostile to XMODEM.

The XMODEM protocol extensions and YMODEM Batch address some of these weaknesses while maintaining most of XMODEM's simplicity.

YMODEM is supported by the public domain programs YAM (CP/M), YAM(CP/M-86), YAM(CCPM-86), IMP (CP/M), KMD (CP/M), rz/sz (Unix, Xenix, VMS, Berkeley Unix, Venix, Xenix, Coherent, IDRIS, Regulus). Commercial implementations include MIRROR, and Professional-YAM.[1] Communications programs supporting these extensions have been in use since 1981.

The 1k block length (XMODEM-1k) described below may be used in conjunction with YMODEM Batch Protocol, or with single file transfers identical to the XMODEM/CRC protocol except for minimal changes to support 1k blocks.

Another extension is the YMODEM-g protocol. YMODEM-g provides batch transfers with maximum throughput when used with end to end error correcting media, such as X.PC and error correcting modems, including 9600 bps units by TeleBit, U.S.Robotics, Hayes, Electronic Vaults, Data Race, and others.

To complete this tome, edited versions of Ward Christensen's original protocol document and John Byrns's CRC-16 document are included for reference.

References to the MODEM or MODEM7 protocol have been changed to XMODEM to accommodate the vernacular. In Australia, it is properly called the Christensen Protocol.

## 3.1 Some Messages from the Pioneer

```
#: 130940 S0/Communications 25-Apr-85  18:38:47
Sb: my protocol
Fm: Ward Christensen 76703,302 [2]
To: all
```

Be aware the article[3] DID quote me correctly in terms of the phrases like "not robust", etc.

It was a quick hack I threw together, very unplanned (like everything I do), to satisfy a personal need to communicate with "some other" people.

ONLY the fact that it was done in 8/77, and that I put it in the public domain immediately, made it become the standard that it is.

---

1. Available for IBM PC,XT,AT, Unix and Xenix

2. Edited for typesetting appearance

3. Infoworld April 29 p. 16

I think its time for me to

(1) document it; (people call me and say "my product is going to include it - what can I 'reference'", or "I'm writing a paper on it, what do I put in the bibliography") and

(2) propose an "incremental extension" to it, which might take "exactly" the form of Chuck Forsberg's YAM protocol. He wrote YAM in C for CP/M and put it in the public domain, and wrote a batch protocol for Unix[4] called rb and sb (receive batch, send batch), which was basically XMODEM with
   (a) a record 0 containing filename date time and size
   (b) a 1K block size option
   (c) CRC-16.

He did some clever programming to detect false ACK or EOT, but basically left them the same.

People who suggest I make SIGNIFICANT changes to the protocol, such as "full duplex", "multiple outstanding blocks", "multiple destinations", etc etc don't understand that the incredible simplicity of the protocol is one of the reasons it survived to this day in as many machines and programs as it may be found in!

Consider the PC-NET group back in '77 or so - documenting to beat the band - THEY had a protocol, but it was "extremely complex", because it tried to be "all things to all people" - i.e. send binary files on a 7-bit system, etc. I was not that "benevolent". I (emphasize > I < ) had an 8-bit UART, so "my protocol was an 8-bit protocol", and I would just say "sorry" to people who were held back by 7-bit limitations. ...

Block size: Chuck Forsberg created an extension of my protocol, called YAM, which is also supported via his public domain programs for UNIX called rb and sb - receive batch and send batch. They cleverly send a "block 0" which contains the filename, date, time, and size. Unfortunately, its UNIX style, and is a bit weird[5] - octal numbers, etc. BUT, it is a nice way to overcome the kludgy "echo the chars of the name" introduced with MODEM7. Further, chuck uses CRC-16 and optional 1K blocks. Thus the record 0, 1K, and CRC, make it a "pretty slick new protocol" which is not significantly different from my own.

Also, there is a catchy name - YMODEM. That means to some that it is the "next thing after XMODEM", and to others that it is the Y(am)MODEM

___

4. VAX/VMS versions of these programs are also available.

5. The file length, time, and file mode are optional. The pathname and file length may be sent alone if desired.

Chapter 3

protocol.  I don't want to emphasize that too much - out of fear that
other mfgrs might think it is a "competitive" protocol, rather than an
"unaffiliated" protocol.  Chuck is currently selling a much-enhanced
version of his CP/M-80 C program YAM, calling it Professional Yam, and its
for the PC - I'm using it right now.  VERY slick!  32K capture buffer,
script, scrolling, previously captured text search, plus built-in commands
for just about everything - directory (sorted every which way), XMODEM,
YMODEM, KERMIT, and ASCII file upload/download, etc.  You can program it
to "behave" with most any system - for example when trying a number for
CIS it detects the "busy" string back from the modem and substitutes a
diff phone # into the dialing string and branches back to try it.

Chapter 3

# 4. XMODEM PROTOCOL ENHANCEMENTS

This chapter discusses the protocol extensions to Ward Christensen's 1982 XMODEM protocol description document.

The original document recommends the user be asked whether to continue trying or abort after 10 retries. Most programs no longer ask the operator whether he wishes to keep retrying. Virtually all correctable errors are corrected within the first few retransmissions. If the line is so bad that ten attempts are insufficient, there is a significant danger of undetected errors. If the connection is that bad, it's better to redial for a better connection, or mail a floppy disk.

## 4.1 Graceful Abort

The YAM and Professional-YAM X/YMODEM routines recognize a sequence of two consecutive CAN (Hex 18) characters without modem errors (overrun, framing, etc.) as a transfer abort command. This sequence is recognized when is waiting for the beginning of a block or for an acknowledgement to a block that has been sent. The check for two consecutive CAN characters reduces the number of transfers aborted by line hits. YAM sends eight CAN characters when it aborts an XMODEM, YMODEM, or ZMODEM protocol file transfer. Pro-YAM then sends eight backspaces to delete the CAN characters from the remote's keyboard input buffer, in case the remote had already aborted the transfer and was awaiting a keyboarded command.

## 4.2 CRC-16 Option

The XMODEM protocol uses an optional two character CRC-16 instead of the one character arithmetic checksum used by the original protocol and by most commercial implementations. CRC-16 guarantees detection of all single and double bit errors, all errors with an odd number of error bits, all burst errors of length 16 or less, 99.9969% of all 17-bit error bursts, and 99.9984 per cent of all possible longer error bursts. By contrast, a double bit error, or a burst error of 9 bits or more can sneak past the XMODEM protocol arithmetic checksum.

The XMODEM/CRC protocol is similar to the XMODEM protocol, except that the receiver specifies CRC-16 by sending C (Hex 43) instead of NAK when requesting the FIRST block. A two byte CRC is sent in place of the one byte arithmetic checksum.

YAM's c option to the r command enables CRC-16 in single file reception, corresponding to the original implementation in the MODEM7 series programs. This remains the default because many commercial communications programs and bulletin board systems still do not support CRC-16, especially those written in Basic or Pascal.

XMODEM protocol with CRC is accurate provided both sender and receiver

both report a successful transmission. The protocol is robust in the presence of characters lost by buffer overloading on timesharing systems.

The single character ACK/NAK responses generated by the receiving program adapt well to split speed modems, where the reverse channel is limited to ten per cent or less of the main channel's speed.

XMODEM and YMODEM are half duplex protocols which do not attempt to transmit information and control signals in both directions at the same time. This avoids buffer overrun problems that have been reported by users attempting to exploit full duplex asynchronous file transfer protocols such as Blast.

Professional-YAM adds several proprietary logic enhancements to XMODEM's error detection and recovery. These compatible enhancements eliminate most of the bad file transfers other programs make when using the XMODEM protocol under less than ideal conditions.

## 4.3 XMODEM-1k 1024 Byte Block

Disappointing throughput downloading from Unix with YMODEM[1] lead to the development of 1024 byte blocks in 1982. 1024 byte blocks reduce the effect of delays from timesharing systems, modems, and packet switched networks on throughput by 87.5 per cent in addition to decreasing XMODEM's 3 per cent overhead (block number, CRC, etc.).

Some environments cannot accept 1024 byte bursts, including some networks and minicomputer ports. The longer block length should be an option.

The choice to use 1024 byte blocks is expressed to the sending program on its command line or selection menu.[2] 1024 byte blocks improve throughput in many applications.

An STX (02) replaces the SOH (01) at the beginning of the transmitted block to notify the receiver of the longer block length. The transmitted block contains 1024 bytes of data. The receiver should be able to accept any mixture of 128 and 1024 byte blocks. The block number (in the second and third bytes of the block) is incremented by one for each block regardless of the block length.

The sender must not change between 128 and 1024 byte block lengths if it has not received a valid ACK for the current block. Failure to observe

1. The name hadn't been coined yet, but the protocol was the same.

2. See "KMD/IMP Exceptions to YMODEM" below.

this restriction allows transmission errors to pass undetected.

If 1024 byte blocks are being used, it is possible for a file to "grow" up to the next multiple of 1024 bytes. This does not waste disk space if the allocation granularity is 1k or greater. With YMODEM batch transmission, the optional file length transmitted in the file name block allows the receiver to discard the padding, preserving the exact file length and contents.

1024 byte blocks may be used with batch file transmission or with single file transmission. CRC-16 should be used with the k option to preserve data integrity over phone lines. If a program wishes to enforce this recommendation, it should cancel the transfer, then issue an informative diagnostic message if the receiver requests checksum instead of CRC-16.

Under no circumstances may a sending program use CRC-16 unless the receiver commands CRC-16.

Figure 1. XMODEM-1k Blocks

```
SENDER                                  RECEIVER
                                        "sx -k foo.bar"
"foo.bar open x.x minutes"
                                        C
STX 01 FE Data[1024] CRC CRC
                                        ACK
STX 02 FD Data[1024] CRC CRC
                                        ACK
STX 03 FC Data[1000] CPMEOF[24] CRC CRC
                                        ACK
EOT
                                        ACK
```

Figure 2. Mixed 1024 and 128 byte Blocks

```
SENDER                                  RECEIVER
                                        "sx -k foo.bar"
"foo.bar open x.x minutes"
                                        C
STX 01 FE Data[1024] CRC CRC
                                        ACK
STX 02 FD Data[1024] CRC CRC
                                        ACK
SOH 03 FC Data[128] CRC CRC
                                        ACK
SOH 04 FB Data[100] CPMEOF[28] CRC CRC
                                        ACK
EOT
                                        ACK
```

## 5. YMODEM Batch File Transmission

The YMODEM Batch protocol is an extension to the XMODEM/CRC protocol that allows 0 or more files to be transmitted with a single command. (Zero files may be sent if none of the requested files is accessible.) The design approach of the YMODEM Batch protocol is to use the normal routines for sending and receiving XMODEM blocks in a layered fashion similar to packet switching methods.

Why was it necessary to design a new batch protocol when one already existed in MODEM7?[1] The batch file mode used by MODEM7 is unsuitable because it does not permit full pathnames, file length, file date, or other attribute information to be transmitted. Such a restrictive design, hastily implemented with only CP/M in mind, would not have permitted extensions to current areas of personal computing such as Unix, DOS, and object oriented systems. In addition, the MODEM7 batch file mode is somewhat susceptible to transmission impairments.

As in the case of single a file transfer, the receiver initiates batch file transmission by sending a "C" character (for CRC-16).

The sender opens the first file and sends block number 0 with the following information.[2]

Only the pathname (file name) part is required for batch transfers.

To maintain upwards compatibility, all unused bytes in block 0 must be set to null.

Pathname The pathname (conventionally, the file name) is sent as a null terminated ASCII string. This is the filename format used by the handle oriented MSDOS(TM) functions and C library fopen functions. An assembly language example follows:

```
DB      'foo.bar',0
```

No spaces are included in the pathname. Normally only the file name stem (no directory prefix) is transmitted unless the sender has selected YAM's f option to send the full pathname. The source drive (A:, B:, etc.) is not sent.

Filename Considerations:

1. The MODEM7 batch protocol transmitted CP/M FCB bytes f1...f8 and t1...t3 one character at a time. The receiver echoed these bytes as received, one at a time.

2. Only the data part of the block is described here.

+ File names are forced to lower case unless the sending system supports upper/lower case file names. This is a convenience for users of systems (such as Unix) which store filenames in upper and lower case.

  + The receiver should accommodate file names in lower and upper case.

  + When transmitting files between different operating systems, file names must be acceptable to both the sender and receiving operating systems.

If directories are included, they are delimited by /; i.e., "subdir/foo" is acceptable, "subdir\foo" is not.

Length The file length and each of the succeeding fields are optional.[3] The length field is stored in the block as a decimal string counting the number of data bytes in the file.  The file length does not include any CPMEOF (^Z) or other garbage characters used to pad the last block.

If the file being transmitted is growing during transmission, the length field should be set to at least the final expected file length, or not sent.

The receiver stores the specified number of characters, discarding any padding added by the sender to fill up the last block.

Modification Date The mod date is optional, and the filename and length may be sent without requiring the mod date to be sent.

Iff the modification date is sent, a single space separates the modification date from the file length.

The mod date is sent as an octal number giving the time the contents of the file were last changed, measured in seconds from Jan 1 1970 Universal Coordinated Time (GMT).  A date of 0 implies the modification date is unknown and should be left as the date the file is received.

This standard format was chosen to eliminate ambiguities arising from transfers between different time zones.

---

3. Fields may not be skipped.

Mode Iff the file mode is sent, a single space separates the file mode from the modification date.  The file mode is stored as an octal string.  Unless the file originated from a Unix system, the file mode is set to 0.  rb(1) checks the file mode for the 0x8000 bit which indicates a Unix type regular file.  Files with the 0x8000 bit set are assumed to have been sent from another Unix (or similar) system which uses the same file conventions. Such files are not translated in any way.

Serial Number Iff the serial number is sent, a single space separates the serial number from the file mode. The serial number of the transmitting program is stored as an octal string. Programs which do not have a serial number should omit this field, or set it to 0. The receiver's use of this field is optional.

Other Fields YMODEM was designed to allow additional header fields to be added as above without creating compatibility problems with older YMODEM programs. Please contact Omen Technology if other fields are needed for special application requirements.

The rest of the block is set to nulls. This is essential to preserve upward compatibility.[4]

If the filename block is received with a CRC or other error, a retransmission is requested. After the filename block has been received, it is ACK'ed if the write open is successful. If the file cannot be opened for writing, the receiver cancels the transfer with CAN characters as described above.

The receiver then initiates transfer of the file contents with a "C" character, according to the standard XMODEM/CRC protocol.

After the file contents and XMODEM EOT have been transmitted and acknowledged, the receiver again asks for the next pathname.

Transmission of a null pathname terminates batch file transmission.

Note that transmission of no files is not necessarily an error. This is possible if none of the files requested of the sender could be opened for reading.

---

4. If, perchance, this information extends beyond 128 bytes (possible with Unix 4.2 BSD extended file names), the block should be sent as a 1k block as described above.

Most YMODEM receivers request CRC-16 by default.

The Unix programs sz(1) and rz(1) included in the source code file RZSZ.ZOO should answer other questions about YMODEM batch protocol.

Figure 3. YMODEM Batch Transmission Session (1 file)

```
          SENDER                                      RECEIVER
                                                      "sb foo.*"
          "sending in batch mode etc."
                                                      C (command:rb)
          SOH 00 FF foo.c NUL[123] CRC CRC
                                                      ACK
                                                      C
          SOH 01 FE Data[128] CRC CRC
                                                      ACK
          SOH 02 FC Data[128] CRC CRC
                                                      ACK
          SOH 03 FB Data[100] CPMEOF[28] CRC CRC
                                                      ACK
          EOT
                                                      NAK
          EOT
                                                      ACK
                                                      C
          SOH 00 FF NUL[128] CRC CRC
                                                      ACK
```

Figure 7. YMODEM Header Information and Features

| Program | Length | Date | Mode | S/N | 1k-Blk | YMODEM-g |
|---|---|---|---|---|---|---|
| Unix rz/sz | yes | yes | yes | no | yes | sb only |
| VMS rb/sb | yes | no | no | no | yes | no |
| Pro-YAM | yes | yes | no | yes | yes | yes |
| CP/M YAM | no | no | no | no | yes | no |
| KMD/IMP | ? | no | no | no | yes | no |

## 5.1 KMD/IMP Exceptions to YMODEM

KMD and IMP use a "CK" character sequence emitted by the receiver to trigger the use of 1024 byte blocks as an alternative to specifying this option to the sending program. This two character sequence generally works well on single process micros in direct communication, provided the programs rigorously adhere to all the XMODEM recommendations included

Figure 4. YMODEM Batch Transmission Session (2 files)

```
       SENDER                                      RECEIVER
                                                   "sb foo.c baz.c"
       "sending in batch mode etc."
                                                   C (command:rb)
       SOH 00 FF foo.c NUL[123] CRC CRC
```

```
                                               ACK
                                               C
       SOH 01 FE Data[128] CRC CRC
                                               ACK
       SOH 02 FC Data[128] CRC CRC
                                               ACK
       SOH 03 FB Data[100] CPMEOF[28] CRC CRC
                                               ACK
       EOT
                                               NAK
       EOT
                                               ACK
                                               C
       SOH 00 FF baz.c NUL[123] CRC CRC
                                               ACK
                                               C
       SOH 01 FB Data[100] CPMEOF[28] CRC CRC
                                               ACK
       EOT
                                               NAK
       EOT
                                               ACK
                                               C
       SOH 00 FF NUL[128] CRC CRC
                                               ACK
```

Figure 5.  YMODEM Batch Transmission Session-1k Blocks

```
        SENDER                                  RECEIVER
                                                "sb -k foo.*"
        "sending in batch mode etc."
                                                C (command:rb)
        SOH 00 FF foo.c NUL[123] CRC CRC
                                                ACK
                                                C
        STX 01 FD Data[1024] CRC CRC
                                                ACK
        SOH 02 FC Data[128] CRC CRC
                                                ACK
        SOH 03 FB Data[100] CPMEOF[28] CRC CRC
                                                ACK
        EOT
                                                NAK
        EOT
```

```
                                                ACK
                                                C
        SOH 00 FF NUL[128] CRC CRC
                                                ACK
```

Figure 6.  YMODEM Filename block transmitted by sz

```
       -rw-r--r--  6347 Jun 17 1984 20:34 bbcsched.txt
       00 0100FF62 62637363 6865642E 74787400   |...bbcsched.txt.|
       10 36333437 20333331 34373432 35313320   |6347 3314742513 |
       20 31303036 34340000 00000000 00000000   |100644..........|
       30 00000000 00000000 00000000 00000000
       40 00000000 00000000 00000000 00000000
       50 00000000 00000000 00000000 00000000
       60 00000000 00000000 00000000 00000000
       70 00000000 00000000 00000000 00000000
       80 000000CA 56
```

herein. Programs with marginal XMODEM implementations do not fare so well. Timesharing systems and packet switched networks can separate the successive characters, rendering this method unreliable.

Sending programs may detect the CK sequence if the operating enviornment does not preclude reliable implementation.

Instead of the standard YMODEM file length in decimal, KMD and IMP transmit the CP/M record count in the last two bytes of the header block.

## 6. YMODEM-g File Transmission

Developing technology is providing phone line data transmission at ever higher speeds using very specialized techniques. These high speed modems, as well as session protocols such as X.PC, provide high speed, nearly error free communications at the expense of considerably increased delay time.

This delay time is moderate compared to human interactions, but it cripples the throughput of most error correcting protocols.

The g option to YMODEM has proven effective under these circumstances. The g option is driven by the receiver, which initiates the batch transfer by transmitting a G instead of C. When the sender recognizes the G, it bypasses the usual wait for an ACK to each transmitted block, sending succeeding blocks at full speed, subject to XOFF/XON or other flow control exerted by the medium.

The sender expects an inital G to initiate the transmission of a particular file, and also expects an ACK on the EOT sent at the end of each file. This synchronization allows the receiver time to open and

close files as necessary.

If an error is detected in a YMODEM-g transfer, the receiver aborts the transfer with the multiple CAN abort sequence. The ZMODEM protocol should be used in applications that require both streaming throughput and error recovery.

Figure 8. YMODEM-g Transmission Session

```
        SENDER                                  RECEIVER
                                                "sb foo.*"
        "sending in batch mode etc..."
                                                G (command:rb -g)
        SOH 00 FF foo.c NUL[123] CRC CRC
                                                G
        SOH 01 FE Data[128] CRC CRC
        STX 02 FD Data[1024] CRC CRC
        SOH 03 FC Data[128] CRC CRC
        SOH 04 FB Data[100] CPMEOF[28] CRC CRC
        EOT
                                                ACK
                                                G
        SOH 00 FF NUL[128] CRC CRC
```

## 7. XMODEM PROTOCOL OVERVIEW

8/9/82 by Ward Christensen.

I will maintain a master copy of this. Please pass on changes or suggestions via CBBS/Chicago at (312) 545-8086, CBBS/CPMUG (312) 849-1132 or by voice at (312) 849-6279.

### 7.1 Definitions

```
   01H
   04H
   06H
   15H
   18H
     43H
```

## 7.2  Transmission Medium Level Protocol

Asynchronous, 8 data bits, no parity, one stop bit.

The protocol imposes no restrictions on the contents of the data being transmitted.  No control characters are looked for in the 128-byte data messages.  Absolutely any kind of data may be sent - binary, ASCII, etc. The protocol has not formally been adopted to a 7-bit environment for the transmission of ASCII-only (or unpacked-hex) data , although it could be simply by having both ends agree to AND the protocol-dependent data with 7F hex before validating it.  I specifically am referring to the checksum, and the block numbers and their ones- complement.

Those wishing to maintain compatibility of the CP/M file structure, i.e. to allow modemming ASCII files to or from CP/M systems should follow this data format:

+ ASCII tabs used (09H); tabs set every 8.

+ Lines terminated by CR/LF (0DH 0AH)

+ End-of-file indicated by ^Z, 1AH.  (one or more)

+ Data is variable length, i.e. should be considered a continuous stream of data bytes, broken into 128-byte chunks purely for the purpose of transmission.

+ A CP/M "peculiarity": If the data ends exactly on a 128-byte boundary, i.e. CR in 127, and LF in 128, a subsequent sector containing the ^Z EOF character(s) is optional, but is preferred. Some utilities or user programs still do not handle EOF without ^Zs.

+ The last block sent is no different from others, i.e.  there is no "short block".

Figure 9.  XMODEM Message Block Level Protocol

```
Each block of the transfer looks like:
      <255-blk #><--128 data bytes-->
in which:
         = 01 hex
       = binary number, starts at 01 increments by 1, and
               wraps 0FFH to 00H (not to 01)
<255-blk #>   = blk # after going thru 8080 "CMA" instr, i.e.
```

```
                 each bit complemented in the 8-bit block number.
                 Formally, this is the "ones complement".
        = the sum of the data bytes only.  Toss any carry.
```

7.3  File Level Protocol

7.3.1  Common_to_Both_Sender_and_Receiver
All errors are retried 10 times.  For versions running with an operator (i.e. NOT with XMODEM), a message is typed after 10 errors asking the operator whether to "retry or quit".

Some versions of the protocol use , ASCII ^X, to cancel transmission. This was never adopted as a standard, as having a single "abort" character makes the transmission susceptible to false termination due to an or  being corrupted into a  and aborting transmission.

The protocol may be considered "receiver driven", that is, the sender need not automatically re-transmit, although it does in the current implementations.

7.3.2  Receive_Program_Considerations
The receiver has a 10-second timeout.  It sends a  every time it times out.  The receiver's first timeout, which sends a , signals the transmitter to start.  Optionally, the receiver could send a immediately, in case the sender was ready.  This would save the initial 10 second timeout.  However, the receiver MUST continue to timeout every 10 seconds in case the sender wasn't ready.

Once into a receiving a block, the receiver goes into a one-second timeout for each character and the checksum.  If the receiver wishes to  a block for any reason (invalid header, timeout receiving data), it must wait for the line to clear.  See "programming tips" for ideas

Synchronizing:  If a valid block number is received, it will be: 1) the expected one, in which case everything is fine; or 2) a repeat of the previously received block.  This should be considered OK, and only indicates that the receivers  got glitched, and the sender re-transmitted; 3) any other block number indicates a fatal loss of synchronization, such as the rare case of the sender getting a line-glitch

that looked like an .  Abort the transmission, sending a

7.3.3  Sending_program_considerations
While waiting for transmission to begin, the sender has only a single very long timeout, say one minute.  In the current protocol, the sender has a 10 second timeout before retrying.  I suggest NOT doing this, and letting the protocol be completely receiver-driven.  This will be compatible with existing programs.

When the sender has no more data, it sends an , and awaits an , resending the  if it doesn't get one.  Again, the protocol could be

receiver-driven, with the sender only having the high-level 1-minute
timeout to abort.

Here is a sample of the data flow, sending a 3-block message. It includes
the two most common line hits - a garbaged block, and an  reply
getting garbaged.  represents the checksum byte.

Figure 10. Data flow including Error Recovery

```
SENDER                                  RECEIVER
                              times out after 10 seconds,
                              <---
 01 FE -data-        --->
                              <---
 02 FD -data- xx         --->       (data gets line hit)
                              <---
 02 FD -data- xx         --->
                              <---
 03 FC -data- xx         --->
(ack gets garbaged)           <---
 03 FC -data- xx         --->
                         --->
                              <---
                         --->
                              <---
(finished)
```

## 7.4 Programming Tips

+ The character-receive subroutine should be called with a parameter specifying the number of seconds to wait. The receiver should first call it with a time of 10, then  and try again, 10 times.

  After receiving the , the receiver should call the character receive subroutine with a 1-second timeout, for the remainder of the message and the . Since they are sent as a continuous stream, timing out of this implies a serious like glitch that caused, say, 127 characters to be seen instead of 128.

+ When the receiver wishes to , it should call a "PURGE" subroutine, to wait for the line to clear. Recall the sender tosses any characters in its UART buffer immediately upon completing sending a block, to ensure no glitches were mis- interpreted.

  The most common technique is for "PURGE" to call the character receive subroutine, specifying a 1-second timeout,[1] and looping back to PURGE until a timeout occurs. The  is then sent, ensuring the other end will see it.

+ You may wish to add code recommended by John Mahr to your character receive routine - to set an error flag if the UART shows framing error, or overrun. This will help catch a few more glitches - the most common of which is a hit in the high bits of the byte in two consecutive bytes. The comes out OK since counting in 1-byte produces the same result of adding 80H + 80H as with adding 00H + 00H.

---

1. These times should be adjusted for use with timesharing systems.

## 8. XMODEM/CRC Overview

Original 1/13/85 by John Byrns -- CRC option.

Please pass on any reports of errors in this document or suggestions for improvement to me via Ward's/CBBS at (312) 849-1132, or by voice at (312) 885-1105.

The CRC used in the Modem Protocol is an alternate form of block check which provides more robust error detection than the original checksum. Andrew S. Tanenbaum says in his book, Computer Networks, that the CRC-CCITT used by the Modem Protocol will detect all single and double bit errors, all errors with an odd number of bits, all burst errors of length 16 or less, 99.997% of 17-bit error bursts, and 99.998% of 18-bit and longer bursts.[1]

The changes to the Modem Protocol to replace the checksum with the CRC are straight forward. If that were all that we did we would not be able to communicate between a program using the old checksum protocol and one using the new CRC protocol. An initial handshake was added to solve this problem. The handshake allows a receiving program with CRC capability to determine whether the sending program supports the CRC option, and to switch it to CRC mode if it does. This handshake is designed so that it will work properly with programs which implement only the original protocol. A description of this handshake is presented in section 10.

Figure 11. Message Block Level Protocol, CRC mode

```
Each block of the transfer in CRC mode looks like:
     <255-blk #><--128 data bytes-->
in which:
        = 01 hex
      = binary number, starts at 01 increments by 1, and
               wraps 0FFH to 00H (not to 01)
<255-blk #>  = ones complement of blk #.
     = byte containing the 8 hi order coefficients of the CRC.
     = byte containing the 8 lo order coefficients of the CRC.
```

## 8.1 CRC Calculation

### 8.1.1 Formal_Definition

To calculate the 16 bit CRC the message bits are considered to be the coefficients of a polynomial. This message polynomial is first multiplied by X^16 and then divided by the generator polynomial (X^16 + X^12 + X^5 +

1. This reliability figure is misleading because XMODEM's critical supervisory functions are not protected by this CRC.

1) using modulo two arithmetic. The remainder left after the division is the desired CRC. Since a message block in the Modem Protocol is 128 bytes or 1024 bits, the message polynomial will be of order X^1023. The hi order bit of the first byte of the message block is the coefficient of X^1023 in the message polynomial. The lo order bit of the last byte of the message block is the coefficient of X^0 in the message polynomial.

Figure 12. Example of CRC Calculation written in C

The following XMODEM crc routine is taken from "rbsb.c". Please refer to the source code for these programs (contained in RZSZ.ZOO) for usage. A fast table driven version is also included in this file.

```
/* update CRC */
unsigned short
```

```
updcrc(c, crc)
register c;
register unsigned crc;
{
        register count;

        for (count=8; --count>=0;) {
                if (crc & 0x8000) {
                        crc <<= 1;
                        crc += (((c<<=1) & 0400)  !=  0);
                        crc ^= 0x1021;
                }
                else {
                        crc <<= 1;
                        crc += (((c<<=1) & 0400)  !=  0);
                }
        }
        return crc;
}
```

## 8.2  CRC File Level Protocol Changes

### 8.2.1  Common_to_Both_Sender_and_Receiver

The only change to the File Level Protocol for the CRC option is the initial handshake which is used to determine if both the sending and the receiving programs support the CRC mode. All Modem Programs should support the checksum mode for compatibility with older versions.  A receiving program that wishes to receive in CRC mode implements the mode setting handshake by sending a  in place of the initial .  If the sending program supports CRC mode it will recognize the  and will set itself into CRC mode, and respond by sending the first block as if a  had been received. If the sending program does not support CRC mode it will not respond to the  at all. After the receiver has sent the  it will wait up to 3 seconds for the  that starts the first block. If it receives a  within 3 seconds it will assume the sender supports CRC mode and will proceed with the file exchange in CRC mode. If no  is

received within 3 seconds the receiver will switch to checksum mode, send a , and proceed in checksum mode. If the receiver wishes to use checksum mode it should send an initial  and the sending program should respond to the  as defined in the original Modem Protocol. After the mode has been set by the initial  or  the protocol follows the original Modem Protocol and is identical whether the checksum or CRC is being used.

### 8.2.2  Receive_Program_Considerations

There are at least 4 things that can go wrong with the mode setting handshake.

1. the initial  can be garbled or lost.

2. the initial  can be garbled.

```
  3.  the initial  can be changed to a .

  4.  the initial  from a receiver which wants to receive in checksum
      can be changed to a .

The first problem can be solved if the receiver sends a second  after
it times out the first time. This process can be repeated several times.
It must not be repeated too many times before sending a  and
switching to checksum mode or a sending program without CRC support may
time out and abort. Repeating the  will also fix the second problem if
the sending program cooperates by responding as if a  were received
instead of ignoring the extra .

It is possible to fix problems 3 and 4 but probably not worth the trouble
since they will occur very infrequently. They could be fixed by switching
modes in either the sending or the receiving program after a large number
of successive s. This solution would risk other problems however.


8.2.3  Sending_Program_Considerations
The sending program should start in the checksum mode. This will insure
compatibility with checksum only receiving programs. Anytime a  is
received before the first  or  the sending program should set
itself into CRC mode and respond as if a  were received. The sender
should respond to additional s as if they were s until the first
 is received. This will assist the receiving program in determining
the correct mode when the  is lost or garbled. After the first
is received the sending program should ignore s.
```

```
8.3  Data Flow Examples with CRC Option

Here is a data flow example for the case where the receiver requests
transmission in the CRC mode but the sender does not support the CRC
option. This example also includes various transmission errors.
represents the checksum byte.

      Figure 13.  Data Flow: Receiver has CRC Option, Sender Doesn't

SENDER                                             RECEIVER
                        <---
                                times out after 3 seconds,
                        <---
                                times out after 3 seconds,
                        <---
                                times out after 3 seconds,
                        <---
```

```
                                times out after 3 seconds,
                        <---
 01 FE -data-  --->
                        <---
 02 FD -data-  --->        (data gets line hit)
                        <---
 02 FD -data-  --->
                        <---
 03 FC -data-  --->
   (ack gets garbaged)  <---
                                times out after 10 seconds,
                        <---
 03 FC -data-  --->
                        <---
                   --->
                        <---
```

Here is a data flow example for the case where the receiver requests
transmission in the CRC mode and the sender supports the CRC option.  This
example also includes various transmission errors.   represents the
2 CRC bytes.

           Figure 14.  Receiver and Sender Both have CRC Option

```
SENDER                                       RECEIVER
                          <---
 01 FE -data-  --->
                          <---
 02 FD -data-  --->         (data gets line hit)
                          <---
 02 FD -data-  --->
                          <---
 03 FC -data-  --->
(ack gets garbaged)       <---
                                   times out after 10 seconds,
                          <---
 03 FC -data-  --->
                          <---
                     --->
                          <---
```

## 9. MORE INFORMATION

Please contact Omen Technology for troff source files and typeset copies of this document.

### 9.1 TeleGodzilla Bulletin Board

More information may be obtained by calling TeleGodzilla at 503-621-3746. Speed detection is automatic for 1200, 2400 and 19200(Telebit PEP) bps. TrailBlazer modem users may issue the TeleGodzilla trailblazer command to swith to 19200 bps once they have logged in.

Interesting files include RZSZ.ZOO (C source code), YZMODEM.ZOO (Official XMODEM, YMODEM, and ZMODEM protocol descriptions), ZCOMMEXE.ARC, ZCOMMDOC.ARC, and ZCOMMHLP.ARC (PC-DOS shareware comm program with XMODEM, True YMODEM(TM), ZMODEM, Kermit Sliding Windows, Telink, MODEM7 Batch, script language, etc.).

## 9.2  Unix UUCP Access

UUCP sites can obtain the current version of this file with

```
                 uucp omen!/u/caf/public/ymodem.doc /tmp
```

A continually updated list of available files is stored in /usr/spool/uucppublic/FILES.  When retrieving these files with uucp, remember that the destination directory on your system must be writeable by anyone, or the UUCP transfer will fail.

The following L.sys line calls TeleGodzilla (Pro-YAM in host operation). TeleGodzilla determines the incoming speed automatically.

In response to "Name Please:" uucico gives the Pro-YAM "link" command as a user name.  The password (Giznoid) controls access to the Xenix system connected to the IBM PC's other serial port.  Communications between Pro-YAM and Xenix use 9600 bps; YAM converts this to the caller's speed.

Finally, the calling uucico logs in as uucp.

```
omen Any ACU 2400 1-503-621-3746 se:--se: link ord: Giznoid in:--in: uucp
```

## 10.  REVISIONS

6-18-88 Further revised for clarity.  Corrected block numbering in two examples.
10-27-87 Optional fields added for number of files remaining to be sent and total number of bytes remaining to be sent.
10-18-87 Flow control discussion added to 1024 byte block descritpion, minor revisions for clarity per user comments.

8-03-87 Revised for clarity.
5-31-1987 emphasizes minimum requirements for YMODEM, and updates information on accessing files.
9-11-1986 clarifies nomenclature and some minor points.
The April 15 1986 edition clarifies some points concerning CRC calculations and spaces in the header.

## 11.  YMODEM Programs

ZCOMM, A shareware little brother to Professional-YAM, is available as ZCOMMEXE.ARC on TeleGodzilla and other bulletin board systems.  ZCOMM may be used to test YMODEM amd ZMODEM implementations.

Unix programs supporting YMODEM are available on TeleGodzilla in RZSZ.ZOO. This ZOO archive includes a ZCOMM/Pro-YAM/PowerCom script ZUPL.T to upload a bootstrap program MINIRB.C, compile it, and then upload the rest of the files using the compiled MINIRB.  Most Unix like systems are supported, including V7, Xenix, Sys III, 4.2 BSD, SYS V, Idris, Coherent, and Regulus.

A version for VAX-VMS is available in VRBSB.SHQ.

Irv Hoff has added 1k blocks and basic YMODEM batch transfers to the KMD and IMP series programs, which replace the XMODEM and MODEM7/MDM7xx series respectively. Overlays are available for a wide variety of CP/M systems.

Questions about Professional-YAM communications software may be directed to:

```
     Chuck Forsberg
     Omen Technology Inc
     17505-V Sauvie Island Road
     Portland Oregon 97231
     VOICE: 503-621-3406 :VOICE
     Modem: 503-621-3746 Speed: 19200(Telebit PEP),2400,1200,300
     Usenet: ...!tektronix!reed!omen!caf
     CompuServe: 70007,2304
     GEnie: CAF
```

Unlike ZMODEM and Kermit, XMODEM and YMODEM place obstacles in the path of a reliable high performance implementation, evidenced by poor reliability under stress of the industry leaders' XMODEM and YMODEM programs. Omen Technology provides consulting and other services to those wishing to implement XMODEM, YMODEM, and ZMODEM with state of the art features and reliability.

CONTENTS

LIST OF FIGURES