this file was abstracted from the original zmodem.doc after one
night I decided I had enough of all this marketing hype and wanted
to know what zmodem was all about without having to wade hip-deep
through advertising slogans. this text is !(C) MCS 1994

the intended audience of this document are programmers looking for
a compact reference text on how zmodem works and what you should
know to be able to implement conforming zmodem send and receive
software. this is definitely not an 'end user' document and the
examples and data structures are strongly biased towards the 'C'
language. (what ? are there other languages ??)

a lot of work went into the preparation of this document; although
I'm afraid I can't guarantee it's correctness. Jacques Mattheij
or MCS shall not be liable for any damages whatsoever. this file
is provided 'as is' for no fee whatsoever. use is at your own risk.
if these terms are unacceptable to you than delete these files NOW!.

changes relative to zmodem.doc as provided by various sources:

- removal of all historical information
- removal of all plugs relating to omen technology products etc.
  if something is public domain then leave it at that
- removal of all 'poetry'
- removal of all references to xmodem ymodem kermit and so on
- removal of all overstrike typesetting tricks which make this
  file practically uneditable and unviewable
- removal of a lot of irrelevant but nice facts about the wheater and
  some other nice subjects for conversation
- removal of all implementation specific details referring to those
  antiques of telecom 'rz' and 'sz'
- manifest constants added in the text.
- moved footnotes to the appropriate place in the text
- changed number base from octal to hex (welcome to the nineties)
  admitted it looks less ivory tower but it reads a lot easier
  for those who started programming after 1959

some recomendations....

a lot has changed since the original zmodem came out. not so much in the
protocol as well as in the world around it. I would like to de-advertise
several of zmodem's advanced features:

- command sending.
  this is the hackers dream come true. a formalized backdoor into any
  site supporting this file transfer protocol with a relatively easy
  defeated security mechanism. don't implement it; just refuse it.

- file translation.
  the zmodem protocol specification below contains a number of facilities
  to change a file between one os and the next. THIS IS NOT THE PLACE !
  a file TRANSFER protocol should do just that and with a minimum of fuss.
  if you have to start worrying about wheter zmodem just garbled that 4MB
  zip file of yours just downloaded from the states at $1 a minute you're
  ready for some agression. another point may be that the file size will
  change which may give rise to a lot of bugs in zmodem implementations on
  the far side of wherever you are downloading to / from. stick to
  binary. it helps.

- use CRC32 and not CRC16
  apart from the obvious (better error detection) the original CRC16
  implementation is buggy

- do not send the serial number in the ZFILE frame. this is not a
  very useful function

- in many places in the orignal zmodem.doc it was suggested that if
  this or that failed you should step down and attempt a ymodem
  transfer. don't do that ! users know pretty good what they want
  and if they specify a zmodem transfer give them one or give them
  nothing. don't try to be smart. probably something is wrong and it
  is better to exit with some informative message than to go ahead
  with the wrong protocol; apart from keeping your source clean.

- don't use or implement the run length encoding. it is greatly
  hampered by not checking if run length encoding is needed. if
  you specify that ability you're stuck with it. nowadays with
  zip 2.0/unzip 5.0 and better there is absolutely no need for a file
  transfer protocol to busy itself with compression. for $200

```
    you can buy a mnp class 10 modem which does all that and more
    completely transparently without possibly triggering a host of
    bugs in a relatively little exercised part of your hosts software.

  - lzw encoding; see run length encoding.

  - don't use the 'ZXSPARSE' option. chances of finding a system that
    implements it are small and even then 10 : 1 that the file will
    be sent compressed.

  - don't send the 'rz\r' used in the original documentation.
    this is a very nasty way of making a public domain protocol
    dependant on a company. (sorry had to abide by that in the end;
    some implementations trigger auto downloads on this)

In general; keep it simple ! stick to multiple file binary transfers and
try to get some speed out of those boxes. time is spent well on optimizing
and cleaning your source rather than on some obscure seldomely used
feature which will clutter your code.

Whenever I give an example of how not to program in 'C' I refer to the
rz.c and sz.c sources. In more than one way these are true 'classics'.
If you intend to implement zmodem don't bother with these dinosaurs
(to use a popular term); better to write it clean from the start.
```

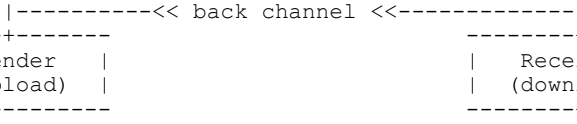
zmodem.doc !copyrighted by MCS 1994 use it anyway you like but don't complain
this file should be accompanied by a readme which contains some background
information on implementing zmodem.

GENERAL

Documentation about the zmodem protocol internals; should be sufficient
to implement a completely functional zmodem protocol suite.

Zmodem is a file transfer protocol that attempts to maximize bandwidth
and minimize transfer times. it is a unidirectional protocol; i.e. the
return channel only transfers control information; no data. either side
may initiate the transfer; but the downloading site may respond to
an initialization frame by auto starting the download software.

Schematically a zmodem file transfer in progress looks like this:



Multiple files may be transferred in one session.

SAMPLE

all zmodem transactions are done using frames. a frame consists
of a header followed by one or more data subpackets.
a typical (simple) zmodem file transfer looks like this :

sender	receiver
ZRQINIT(0)	
	ZRINIT
ZFILE	
	ZRPOS
ZDATA data ...	
ZEOF	
	ZRINIT
ZFIN	
	ZFIN
OO	

zmodem continuously transmits data unless the receiver interrupts
the sender to request retransmission of garbled data.
zmodem in effect uses the entire file as a window.

REQUIREMENTS

zmodem requires an 8 bit transfer medium; but allows encoded packets
for less transparent media.
zmodem escapes network control characters to allow operation with
packet switched networks.

To support full streaming, the transmission path should either assert
flow control or pass full speed transmission without loss of data.
Otherwise the zmodem sender must manage the window size.

zmodem places no constraints on the content files.

LINK ESCAPE ENCODING

zmodem achieves data transparency by extending the 8 bit character set
(256 codes) with escape sequences based on the zmodem data link escape
character ZDLE

Link Escape coding permits variable length data subpackets without the
overhead of a separate byte count. It allows the beginning of frames to
be detected without special timing techniques, facilitating rapid error
recovery.

Link Escape coding does add some overhead. The worst case, a file
consisting entirely of escaped characters, would incur a 50% overhead.

The ZDLE character is special. ZDLE represents a control
sequence of some sort. If a ZDLE character appears in binary data,
it is prefixed with ZDLE, then sent as ZDLEE

5 consecutive CAN characters abort a zmodem session

Since CAN is not used in normal terminal operations, interactive applications and communications programs can monitor the data flow for ZDLE. The following characters can be scanned to detect the ZRQINIT header, the invitation to automatically download commands or files.

Receipt of five successive CAN characters will abort a zmodem session. Eight CAN characters are sent (just to be on the safe side)

The receiving program decodes any sequence of ZDLE followed by a byte with bit 6 set and bit 5 reset (upper case letter, either parity) to the equivalent control character by inverting bit 6. This allows the transmitter to escape any control character that cannot be sent by the communications medium. In addition, the receiver recognizes escapes for 0x7f and 0xff should these characters need to be escaped.

zmodem software escapes ZDLE (0x18), 0x10, 0x90, 0x11, 0x91, 0x13, and 0x93.
If preceded by 0x40 or 0xc0 (@), 0x0d and 0x8d are also escaped to protect the Telenet command escape CR-@-CR. The receiver ignores 0x11, 0x91, 0x13, and 0x93 characters in the data stream.

□

HEADERS

All zmodem frames begin with a header which may be sent in binary or HEX form. Either form of the header contains the same raw information:

- A type byte

- Four bytes of data indicating flags and/or numeric quantities depending on the frame type

the maximum header information length is 16 bytes
the data subpackets following the header are maximum 1024 bytes long.

```
       M         L
FTYPE  F3 F2 F1 F0   (flags frame)

       L         M
FTYPE  P0 P1 P2 P3   (numeric frame)
```

Beware of the catch; flags and numbers are indexed the other way around !

16 BIT CRC BINARY HEADER

A binary header is sent by the sending program to the receiving program. All bytes in a binary header are ZDLE encoded.

A binary header begins with the sequence ZPAD, ZDLE, ZBIN.

0 or more binary data subpackets with 16 bit CRC will follow depending on the frame type.

* ZDLE A TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2

32 BIT CRC BINARY HEADER

A "32 bit CRC" Binary header is similar to a Binary Header, except the ZBIN (A) character is replaced by a ZBIN32 (C) character, and four characters of CRC are sent.

0 or more binary data subpackets with 32 bit CRC will follow depending on the frame type.

* ZDLE C TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2 CRC-3 CRC-4

♀

HEX HEADER

The receiver sends responses in hex headers. The sender also uses hex headers when they are not followed by binary data subpackets.

Hex encoding protects the reverse channel from random control characters. The hex header receiving routine ignores parity.

use of hex headers by the receiving program allows control characters to be used to interrupt the sender when errors are detected.
A HEX header may be used in place of a binary header

wherever convenient.
If a data packet follows a HEX header, it is protected with CRC-16.

A hex header begins with the sequence ZPAD, ZPAD, ZDLE, ZHEX.
The extra ZPAD character allows the sending program to detect
an asynchronous header (indicating an error condition) and then
get the rest of the header with a non-error specific routine.

The type byte, the four position/flag bytes, and the 16 bit CRC
thereof are sent in hex using the character set 01234567890abcdef.
Upper case hex digits are not allowed.
Since this form of hex encoding detects many patterns of errors,
especially missing characters, a hex header with 32 bit CRC has not
been defined.

A carriage return and line feed are sent with HEX headers. The
receive routine expects to see at least one of these characters, two
if the first is CR.

An XON character is appended to all HEX packets except ZACK and ZFIN.
The XON releases the sender from spurious XOFF flow control characters
generated by line noise. XON is not sent after ZACK headers to protect
flow control in streaming situations. XON is not sent after a ZFIN
header to allow proper session cleanup.

0 or more data subpackets will follow depending on the frame type.

* * ZDLE B TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2 CR LF XON

(TYPE, F3...F0, CRC-1, and CRC-2 are each sent as two hex digits.)

BINARY DATA SUBPACKETS

Binary data subpackets immediately follow the associated binary header
packet. A binary data packet contains 0 to 1024 bytes of data.
Recommended length values are 256 bytes below 2400 bps, 512 at
2400 bps, and 1024 above 4800 bps or when the data link is known to
be relatively error free.

No padding is used with binary data subpackets. The data bytes are
ZDLE encoded and transmitted. A ZDLE and frameend are then sent,
followed by two or four ZDLE encoded CRC bytes. The CRC accumulates
the data bytes and frameend.

♀
PROTOCOL TRANSACTION OVERVIEW
---

zmodem timing is receiver driven. The
transmitter should not time out at all, except to abort the program if no
headers are received for an extended period of time, say one minute.

SESSION STARTUP

To start a zmodem file transfer session, the sending program is called
with the names of the desired file(s) and option(s).

Then the sender may send a ZRQINIT header. The ZRQINIT header causes a
previously started receive program to send its ZRINIT header without
delay.

In an interactive or conversational mode, the receiving application
may monitor the data stream for ZDLE. The following characters may be
scanned for B00 indicating a ZRQINIT header, a command to download
data.

The sending program awaits a command from the receiving program to
start file transfers.

In case of garbled data, the sending program can repeat the invitation
to receive a number of times until a session starts.

When the zmodem receive program starts, it immediately sends a ZRINIT
header to initiate zmodem file transfers, or a ZCHALLENGE header to
verify the sending program. The receive program resends its header at
response time (default 10 second) intervals for a suitable period of
time (40 seconds total) before exitting.

If the receiving program receives a ZRQINIT header, it resends the
ZRINIT header. If the sending program receives the ZCHALLENGE header,
it places the data in ZP0...ZP3 in an answering ZACK header.

If the receiving program receives a ZRINIT header, it is an echo

indicating that the sending program is not operational.

Eventually the sending program correctly receives the ZRINIT header.

The sender may then send an optional ZSINIT frame to define the receiving program's Attn sequence, or to specify complete control character escaping. if the receiver specifies the same or higher level of escaping the ZSINIT frame need not be sent unless an Attn sequence is needed.

If the ZSINIT header specifies ESCCTL or ESC8, a HEX header is used, and the receiver activates the specified ESC modes before reading the following data subpacket.

The receiver sends a ZACK header in response, containing 0.

♀

FILE TRANSMISSION

The sender then sends a ZFILE header with zmodem Conversion, Management, and Transport options (see ZFILE header type) followed by a ZCRCW data subpacket containing the file name, file length and modification date.

The receiver examines the file name, length, and date information provided by the sender in the context of the specified transfer options, the current state of its file system(s), and local security requirements. The receiving program should insure the pathname and options are compatible with its operating environment and local security requirements.

The receiver may respond with a ZSKIP header, which makes the sender proceed to the next file (if any) in the batch.

> The receiver has a file with the same name and length, may respond with a ZCRC header with a byte count, which requires the sender to perform a 32 bit CRC on the specified number of bytes in the file and transmit the complement of the CRC in an answering ZCRC header.the crc is initialised to 0xffffffff; a byte count of 0 implies the entire file the receiver uses this information to determine whether to accept the file or skip it. This sequence may be triggered by the ZMCRC Management Option.

A ZRPOS header from the receiver initiates transmission of the file data starting at the offset in the file specified in the ZRPOS header. Normally the receiver specifies the data transfer to begin begin at offset 0 in the file.

> The receiver may start the transfer further down in the file. This allows a file transfer interrupted by a loss of carrier or system crash to be completed on the next connection without requiring the entire file to be retransmitted. If downloading a file from a timesharing system that becomes sluggish, the transfer can be interrupted and resumed later with no loss of data.

The sender sends a ZDATA binary header (with file position) followed by one or more data subpackets.

The receiver compares the file position in the ZDATA header with the number of characters successfully received to the file. If they do not agree, a ZRPOS error response is generated to force the sender to the right position within the file. (if the ZMSPARS option is used the receiver instead seeks to the position given in the ZDATA header)

A data subpacket terminated by ZCRCG and CRC does not elicit a response unless an error is detected; more data subpacket(s) follow immediately.

□

ZCRCQ data subpackets expect a ZACK response with the receiver's file offset if no error, otherwise a ZRPOS response with the last good file offset. Another data subpacket continues immediately. ZCRCQ subpackets are not used if the receiver does not indicate full duplex ability with the CANFDX bit.

ZCRCW data subpackets expect a response before the next frame is sent. If the receiver does not indicate overlapped I/O capability with the CANOVIO bit, or sets a buffer size, the sender uses the ZCRCW to allow the receiver to write its buffer before sending more data.

> A zero length data frame may be used as an idle subpacket to prevent the receiver from timing out in

case data is not immediately available to the sender.

In the absence of fatal error, the sender eventually encounters end of file. If the end of file is encountered within a frame, the frame is closed with a ZCRCE data subpacket which does not elicit a response except in case of error.

The sender sends a ZEOF header with the file ending offset equal to the number of characters in the file. The receiver compares this number with the number of characters received. If the receiver has received all of the file, it closes the file. If the file close was satisfactory, the receiver responds with ZRINIT. If the receiver has not received all the bytes of the file, the receiver ignores the ZEOF because a new ZDATA is coming. If the receiver cannot properly close the file, a ZFERR header is sent.

After all files are processed, any further protocol errors should not prevent the sending program from returning with a success status.

SESSION CLEANUP

The sender closes the session with a ZFIN header. The receiver acknowledges this with its own ZFIN header.

When the sender receives the acknowledging header, it sends two characters, "OO" (Over and Out) and exits.
The receiver waits briefly for the "O" characters, then exits whether they were received or not.

SESSION ABORT SEQUENCE

If the receiver is receiving data in streaming mode, the Attn sequence is executed to interrupt data transmission before the Cancel sequence is sent. The Cancel sequence consists of eight CAN characters and ten backspace characters. zmodem only requires five Cancel characters, the other three are "insurance".

The trailing backspace characters attempt to erase the effects of the CAN characters if they are received by a command interpreter.

```
char ses_abort_seq[] = {
                24,24,24,24,24,24,24,24,8,8,8,8,8,8,8,8,8,8,0
};
```

ATTENTION SEQUENCE

The receiving program sends the Attn sequence whenever it detects an error and needs to interrupt the sending program.

The default Attn string value is empty (no Attn sequence). The receiving program resets Attn to the empty default before each transfer session.

The sender specifies the Attn sequence in its optional ZSINIT frame. The Attn string is terminated with a null.

FRAME TYPES

The numeric values are listed at the end of this file.
Unused bits and unused bytes in the header (ZP0...ZP3) are set to 0.

ZRQINIT

Sent by the sending program, to trigger the receiving program to send its ZRINIT header.
The sending program may
repeat the receive invitation (including ZRQINIT) if a response is not obtained at first.

ZF0 contains ZCOMMAND if the program is attempting to send a command, 0 otherwise.

ZRINIT

Sent by the receiving program. ZF0 and ZF1 contain the bitwise or of the receiver capability flags:

```
                CANCRY          receiver can decrypt
                CANFDX          receiver can send and receive true full duplex
                CANOVIO         receiver can receive data during disk I/O
                CANBRK          receiver can send a break signal
                CANCRY          receiver can decrypt
                CANLZW          receiver can uncompress
                CANFC32         receiver can use 32 bit Frame Check
                ESCCTL          receiver expects ctl chars to be escaped
                ESC8            receiver expects 8th bit to be escaped

        ZP0 and ZP1 contain the size of the receiver's buffer in bytes, or 0
        if nonstop I/O is allowed.

        while all these capabilities are nice in theory most zmodem implem-
        tations balk at anything other than 0,0. i.e. telix starts sending
        35 byte packets when CANFC32 is on.

ZSINIT
------

        The Sender sends flags followed by a binary data subpacket terminated
        with ZCRCW.

                TESCCTL         Transmitter expects ctl chars to be escaped
                TESC8           Transmitter expects 8th bit to be escaped

        The data subpacket contains the null terminated Attn sequence,
        maximum length 32 bytes including the terminating null.

ZACK
----

        Acknowledgment to a ZSINIT frame, ZCHALLENGE header, ZCRCQ or ZCRCW
        data subpacket.  ZP0 to ZP3 contain file offset.  The response to
        ZCHALLENGE contains the same 32 bit number received in the ZCHALLENGE
        header.

□

ZFILE
-----
        This frame denotes the beginning of a file transmission attempt.
        ZF0, ZF1, and ZF2 may contain options. A value of 0 in each of these
        bytes implies no special treatment.  Options specified to the
        receiver override options specified to the sender with the exception
        of ZCBIN.  A ZCBIN from the sender overrides any other Conversion
        Option given to the receiver except ZCRESUM.  A ZCBIN from the
        receiver overrides any other Conversion Option sent by the sender.


        ZF0: CONVERSION OPTION
        ----------------------

                If the receiver does not recognize the Conversion Option, an
                application dependent default conversion may apply.

                ZCBIN "Binary" transfer - inhibit conversion unconditionally

                ZCNL Convert received end of line to local end of line
                     convention.  The supported end of line conventions are
                     CR/LF (most ASCII based operating systems except Unix
                     and Macintosh), and NL (Unix).  Either of these two end
                     of line conventions meet the permissible ASCII
                     definitions for Carriage Return and Line Feed/New Line.
                     Neither the ASCII code nor zmodem ZCNL encompass lines
                     separated only by carriage returns.  Other processing
                     appropriate to ASCII text files and the local operating
                     system may also be applied by the receiver. (filtering
                     RUBOUT NULL Ctrl-Z etc)

                ZCRECOV Recover/Resume interrupted file transfer.  ZCREVOV is
                     also useful for updating a remote copy of a file that
                     grows without resending of old data.  If the destination
                     file exists and is no longer than the source, append to
                     the destination file and start transfer at the offset
                     corresponding to the receiver's end of file.  This
                     option does not apply if the source file is shorter.
                     Files that have been converted (e.g., ZCNL) or subject
                     to a single ended Transport Option cannot have their
                     transfers recovered.

□

        ZF1: Management Option
        ----------------------

                If the receiver does not recognize the Management Option, the
                file should be transferred normally.
```

The ZMSKNOLOC bit instructs the receiver to bypass the current file if the receiver does not have a file with the same name.

Five bits (defined by ZMMASK) define the following set of mutually exclusive management options.

ZMNEWL Transfer file if destination file absent. Otherwise, transfer file overwriting destination if the source file is newer or longer.

ZMCRC Compare the source and destination files. Transfer if file lengths or file polynomials differ.

ZMAPND Append source file contents to the end of the existing destination file (if any).

ZMCLOB Replace existing destination file (if any).

ZMDIFF Transfer file if destination file absent. Otherwise, transfer file overwriting destination if files have different lengths or dates.

ZMPROT Protect destination file by transferring file only if the destination file is absent.

ZMNEW Transfer file if destination file absent. Otherwise, transfer file overwriting destination if the source file is newer.

□

ZF2: TRANSPORT OPTION

If the receiver does not implement the particular transport option, the file is copied without conversion for later processing. better not to use these; see readme

ZTLZW Lempel-Ziv compression. Transmitted data will be identical to that produced by compress 4.0 operating on a computer with VAX byte ordering, using 12 bit encoding.

ZTCRYPT Encryption. An initial null terminated string identifies the key. Details to be determined.

ZTRLE Run Length encoding, Details to be determined.

A ZCRCW data subpacket follows with file name, file length, modification date, and other information described in a later chapter.

ZF3: EXTENDED OPTIONS

The Extended Options are bit encoded.

ZTSPARS Special processing for sparse files, or sender managed selective retransmission. Each file segment is transmitted as a separate frame, where the frames are not necessarily contiguous. The sender should end each segment with a ZCRCW data subpacket and process the expected ZACK to insure no data is lost. ZTSPARS cannot be used with ZCNL.

□

ZSKIP

Sent by the receiver in response to ZFILE, makes the sender skip to the next file.

ZNAK

Indicates last header was garbled. (See also ZRPOS).

ZABORT

Sent by receiver to terminate batch file transfers when requested by the user. Sender responds with a ZFIN sequence. (or ZCOMPL in case of server mode).

ZFIN

Sent by sending program to terminate a zmodem session. Receiver responds with its own ZFIN.

ZRPOS

Sent by receiver to force file transfer to resume at file offset given in ZP0...ZP3.

ZDATA

ZP0...ZP3 contain file offset. One or more data subpackets follow.

ZEOF

Sender reports End of File. ZP0...ZP3 contain the ending file offset.

ZFERR

Error in reading or writing file, protocol equivalent to ZABORT.

ZCRC

Request (receiver) and response (sender) for file polynomial. ZP0...ZP3 contain file polynomial.

ZCHALLENGE

Request sender to echo a random number in ZP0...ZP3 in a ZACK frame. Sent by the receiving program to the sending program to verify that it is connected to an operating program, and was not activated by spurious data or a Trojan Horse message. this is the most simply defeated security system ever invented. don't rely on it and better still don't use or implement it. build your security measures around starting the download at all and disallow explicit path names.

ZCOMPL

Request now completed.

ZCAN

This is a pseudo frame type in response to a Session Abort sequence.

ZFREECNT

Sending program requests a ZACK frame with ZP0...ZP3 containing the number of free bytes on the current file system. A value of 0 represents an indefinite amount of free space.

ZCOMMAND

ZCOMMAND is sent in a binary frame. ZF0 contains 0 or ZCACK1 (see below).

A ZCRCW data subpacket follows, with the ASCII text command string terminated with a NULL character. If the command is intended to be executed by the operating system hosting the receiving program (e.g., "shell escape"), it must have "!" as the first character. Otherwise the command is meant to be executed by the application program which receives the command.

If the receiver detects an illegal or badly formed command, the receiver immediately responds with a ZCOMPL header with an error code in ZP0...ZP3.

If ZF0 contained ZCACK1, the receiver immediately responds with a ZCOMPL header with 0 status.

Otherwise, the receiver responds with a ZCOMPL header when the operation is completed. The exit status of the completed command is stored in ZP0...ZP3. A 0 exit status implies nominal completion of

the command.

If the command causes a file to be transmitted, the command sender will see a ZRQINIT frame from the other computer attempting to send data.

The sender examines ZF0 of the received ZRQINIT header to verify it is not an echo of its own ZRQINIT header. It is illegal for the sending program to command the receiving program to send a command.

If the receiver program does not implement command downloading, it may display the command to the standard error output, then return a ZCOMPL header.

□

ZFILE FRAME FILE INFORMATION SUBPACKET
---

zmodem sends the same file information with the ZFILE frame data

The pathname (file name) field is mandatory. each field after pathname is optional and separated from the previous one by a space. fields may not be skipped. the use of the optional fields is (by definition) optional to the receiver.

PATHNAME

The pathname (conventionally, the file name) is sent as a null terminated ASCII string.
No spaces are included in the pathname.

Filename Considerations

- File names should be translated 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 not, transformations should be applied to make the file names acceptable. If the transformations are unsuccessful, a new file name may be invented be the receiving program.

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

♀

LENGTH

The length field is stored as a decimal string counting the number of data bytes in the file.

The zmodem receiver uses the file length as an estimate only. It may be used to display an estimate of the transmission time, and may be compared with the amount of free disk space. The actual length of the received file is determined by the data transfer. A file may grow after transmission commences, and all the data will be sent.

MODIFICATION DATE

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.

FILE MODE

The file mode is stored as an octal string.
Unless the file originated from a Unix system, the file mode is set to 0.

SERIAL NUMBER

---

set this field to 0.

NUMBER OF FILES REMAINING

This field is coded as a decimal number, and includes the current file. This field is an estimate, and incorrect values must not be allowed to cause loss of data.

NUMBER OF BYTES REMAINING

This field is coded as a decimal number, and includes the current file. This field is an estimate, and incorrect values must not be allowed to cause loss of data.

FILE TYPE

set this field to 0.

The file information is terminated by a null. If only the pathname is sent, the pathname is terminated with two nulls. The length of the file information subpacket, including the trailing null, must not exceed 1024 bytes; a typical length is less than 64 bytes.

STREAMING TECHNIQUES / ERROR RECOVERY

zmodem provides several data streaming methods selected according to the limitations of the sending environment, receiving environment, and transmission channel(s).

WINDOW MANAGEMENT

When sending data through a network, some nodes of the network store data while it is transferred to the receiver. 7000 bytes and more of transient storage have been observed. Such a large amount of storage causes the transmitter to "get ahead" of the receiver.
This condition is not fatal but it does slow error recovery.

To manage the window size, the sending program uses ZCRCQ data subpackets to trigger ZACK headers from the receiver. The returning ZACK headers inform the sender of the receiver's progress. When the window size (current transmitter file offset - last reported receiver file offset) exceeds a specified value, the sender waits for a ZACK packet with a receiver file offset that reduces the window size. ZRPOS and other error packets are handled normally.

FULL STREAMING WITH SAMPLING

If the receiver can overlap serial I/O with disk I/O, and if the sender can sample the reverse channel for the presence of data without having to wait, full streaming can be used with no Attn sequence required. The sender begins data transmission with a ZDATA header and continuous ZCRCG data subpackets. When the receiver detects an error, it executes the Attn sequence and then sends a ZRPOS header with the correct position within the file.

At the end of each transmitted data subpacket, the sender checks for the presence of an error header from the receiver. To do this, the sender samples the reverse data stream for the presence of either a ZPAD or CAN character (using rdchk() or a similar mechanism)
. Flow control characters (if present) are acted upon.

Other characters (indicating line noise) increment a counter which is reset whenever the sender waits for a header from the receiver. If the counter overflows, the sender sends the next data subpacket as ZCRCW, and waits for a response.

ZPAD indicates some sort of error header from the receiver. A CAN suggests the user is attempting to "stop the bubble machine" by keyboarding CAN characters. If one of these characters is seen, an empty ZCRCE data subpacket is sent. Normally, the receiver will have sent an ZRPOS or other error header, which will force the sender to resume transmission at a different address, or take other action. In the unlikely event the ZPAD or CAN character was spurious, the receiver will time out and send a ZRPOS header.
The obvious choice of ZCRCW packet, which would trigger an ZACK from the receiver, is not used because multiple in transit frames could

result if the channel has a long propagation delay.

Then the receiver's response header is read and acted upon; starting with a resinchronize.

A ZRPOS header resets the sender's file offset to the correct position. If possible, the sender should purge its output buffers and/or networks of all unprocessed output data, to minimize the amount of unwanted data the receiver must discard before receiving data starting at the correct file offset. The next transmitted data frame should be a ZCRCW frame followed by a wait to guarantee complete flushing of the network's memory.

If the receiver gets a ZACK header with an address that disagrees with the sender address, it is ignored, and the sender waits for another header. A ZFIN, ZABORT, or TIMEOUT terminates the session; a ZSKIP terminates the processing of this file.

The reverse channel is then sampled for the presence of another header from the receiver (if sampling is possible). if one is detected another error header is read. Otherwise,
transmission resumes at the (possibly reset) file offset with a ZDATA header followed by data subpackets.

♀

FULL STREAMING WITH REVERSE INTERRUPT

The above method cannot be used if the reverse data stream cannot be sampled without entering an I/O wait. An alternate method is to instruct the receiver to interrupt the sending program when an error is detected.

The receiver can interrupt the sender with a control character, break signal, or combination thereof, as specified in the Attn sequence. After executing the Attn sequence, the receiver sends a hex ZRPOS header to force the sender to resend the lost data.

When the sending program responds to this interrupt, it reads a HEX header (normally ZRPOS) from the receiver and takes the action described in the previous section.

FULL STREAMING WITH SLIDING WINDOW

If none of the above methods is applicable, hope is not yet lost. If the sender can buffer responses from the receiver, the sender can use ZCRCQ data subpackets to get ACKs from the receiver without interrupting the transmission of data. After a sufficient number of ZCRCQ data subpackets have been sent, the sender can read one of the headers that should have arrived in its receive interrupt buffer.

A problem with this method is the possibility of wasting an excessive amount of time responding to the receiver's error header. It may be possible to program the receiver's Attn sequence to flush the sender's interrupt buffer before sending the ZRPOS header.

□

SEGMENTED STREAMING

If the receiver cannot overlap serial and disk I/O, it uses the ZRINIT frame to specify a buffer length which the sender will not overflow. The sending program sends a ZCRCW data subpacket and waits for a ZACK header before sending the next segment of the file.

A sufficiently large receiving buffer allows throughput to closely approach that of full streaming. For example, 16kb segmented streaming adds about 3 per cent to full streaming zmodem file transfer times when the round trip delay is five seconds.

□

CONSTANTS

ASCII

SOH	0x01
STX	0x02
EOT	0x04
ENQ	0x05
ACK	0x06
LF	0x0a
CR	0x0d
XON	0x11
XOFF	0x13

```
        NAK                     0x15
        CAN                     0x18

ZMODEM
------

        ZPAD            0x2a    /* pad character; begins frames */
        ZDLE            0x18    /* ctrl-x zmodem escape */
        ZDLEE           0x58    /* escaped ZDLE */

        ZBIN            0x41    /* binary frame indicator (CRC16) */
        ZHEX            0x42    /* hex frame indicator */
        ZBIN32          0x43    /* binary frame indicator (CRC32) */
        ZBINR32         0x44    /* run length encoded binary frame (CRC32) */

        ZVBIN           0x61    /* binary frame indicator (CRC16) */
        ZVHEX           0x62    /* hex frame indicator */
        ZVBIN32         0x63    /* binary frame indicator (CRC32) */
        ZVBINR32        0x64    /* run length encoded binary frame (CRC32) */

        ZRESC           0x7e    /* run length encoding flag / escape character */

FRAME TYPES
-----------

        ZRQINIT         0x00    /* request receive init (s->r) */
        ZRINIT          0x01    /* receive init (r->s) */
        ZSINIT          0x02    /* send init sequence (optional) (s->r) */
        ZACK            0x03    /* ack to ZRQINIT ZRINIT or ZSINIT (s<->r) */
        ZFILE           0x04    /* file name (s->r) */
        ZSKIP           0x05    /* skip this file (r->s) */
        ZNAK            0x06    /* last packet was corrupted (?) */
        ZABORT          0x07    /* abort batch transfers (?) */
        ZFIN            0x08    /* finish session (s<->r) */
        ZRPOS           0x09    /* resume data transmission here (r->s) */
        ZDATA           0x0a    /* data packet(s) follow (s->r) */
        ZEOF            0x0b    /* end of file reached (s->r) */
        ZFERR           0x0c    /* fatal read or write error detected (?) */
        ZCRC            0x0d    /* request for file CRC and response (?) */
        ZCHALLENGE      0x0e    /* security challenge (r->s) */
        ZCOMPL          0x0f    /* request is complete (?) */
        ZCAN            0x10    /* pseudo frame;
                                           other end cancelled session with 5* CAN */
        ZFREECNT        0x11    /* request free bytes on file system (s->r) */
        ZCOMMAND        0x12    /* issue command (s->r) */
        ZSTDERR         0x13    /* output data to stderr (??) */

ZDLE SEQUENCES
--------------

        ZCRCE           0x68    /* CRC next, frame ends, header packet follows */
        ZCRCG           0x69    /* CRC next, frame continues nonstop */
        ZCRCQ           0x6a    /* CRC next, frame continuous, ZACK expected */
        ZCRCW           0x6b    /* CRC next, ZACK expected, end of frame */
        ZRUB0           0x6c    /* translate to rubout 0x7f */
        ZRUB1           0x6d    /* translate to rubout 0xff */

RECEIVER CAPABILITY FLAGS
-------------------------

        CANFDX          0x01    /* Rx can send and receive true full duplex */
        CANOVIO         0x02    /* Rx can receive data during disk I/O */
        CANBRK          0x04    /* Rx can send a break signal */
        CANCRY          0x08    /* Receiver can decrypt */
        CANLZW          0x10    /* Receiver can uncompress */
        CANFC32         0x20    /* Receiver can use 32 bit Frame Check */
        ESCCTL          0x40    /* Receiver expects ctl chars to be escaped
        ESC8            0x80    /* Receiver expects 8th bit to be escaped */
```