XPRZModem

COLLABORATORS			
	TITLE : XPRZModem		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

XPRZModem

1.1 XPRZModem.guide

```
                          XPRZModem.library & XPRzedzap.library
                        Version 3.4
                        Nov 26 1994
      Contents:

            Introduction

            Installation

            Interface

            Options

            Serial Port Settings

            Receiving Files

            Sending Files

            Technical Information

            Future

            Revision History

            To do

            Credits
```

1.2 introduction

```
                                                Introduction

   XPRzModem.library and XPRzedzap.library are Amiga shared libraries (with
```

full source code) which provide ZModem file transfer capabilities to any XPR-compatible communications program. The XPR (eXternal PRotocol) standard describes an interface method which allows various file transfer protocols to be implemented as Amiga shared libraries. These libraries may then be used interchangeably in any compatible communications program. This takes a heavy load off of the comm program author, who no longer has to support dozens of different file transfer protocols in their program in order to make it widely useful and popular. The comm program is also smaller and more efficient as a result, since all those obscure protocols (you know, the ones *you* don't need) are no longer taking up space.

The XPR standard also helps users, who can mix and match their favorite file transfer protocols with their favorite comm programs. And when new protocols are invented, the user simply plugs in a new library, and voila!, it's ready to use. Hopefully, making protocols easy to support will allow more and better comm programs to be written, as authors can concentrate on their programs instead of constantly re-inventing the wheel.

Of course, for all of this wonderful stuff to happen, there needs to be a good selection of these XPR libraries available to the public. It's the classic chicken-and-egg problem; comm program authors won't be motivated to support the XPR standard unless there are a sizable number of protocols available for it. And other programmers won't be motivated to write XPR libraries until there are a sizable number of comm programs which can use them. In an effort to do my bit [B^)] for the Amiga community, which has given me so many neat toys to play with over the past few years, I decided to try and help get the ball rolling.

Hopefully, the early availability of a ZModem library will help stimulate interest in the XPR standard, resulting in better Amiga telecomms for all of us. And by making my source code PD, I hope to help others interested in writing XPR libraries by giving them some serious example code. Also, having ZModem library code readily available to John Q. Hacker should help ensure a steady stream of enhanced ZModem libraries (with enzymes) for all of us to use in the future.

Of course, no discussion of the XPR standard would be complete without giving proper credit to the author,

Willy Langeveld
of the Stanford Linear Accelerator Center. Many thanks are due ↩
him for this

effort. If you have any further questions about the XPR standard, be sure and download the spec; it should be available on BIX (since he's a sysop there), or on most other major systems.

All files in this archive which are not copyrighted are hereby released to the public domain (which they were anyway, by way of not being copyrighted, but I want to make sure YOU realize that). Do as you like with them. Please make lots of copies and distribute them all over the place, and make lots of derivative works, and everything! Heck, you can even publicly perform and/or display this code if you can figure out how...

1.3 installation

Just copy the xprzmodem.library file into your LIBS: directory. All XPR-compatible comm programs should provide a way for you to select which XPR protocol you wish to use, either by giving you a file requester showing LIBS:xpr*.library, or by automatically detecting these libraries and adding them into their menus.

If your intended use is with a Mailer, use xprzedzap.library, selecting appropriate CPU version for your system. You may also use xprzedzap.library in the place of xprzmodem.library in both BBS's and terminal programs.

680x0: For Kickstart 2.x (37.175) or higher, all Processors.
68030: For Kickstart 2.x (37.175) or higher, optimized code for CPU-Type 68030.
68040: For Kickstart 2.x (37.175) or higher, optimized code for CPU-Type 68040.

With this version the Protocol name sent to the Status Display will be one of:

Zmodem	1K blocks standard, non-ftn mode
ZedZap	up to 8K Blocks based upon bps rate, ftn mode
ZedZip	1k blocks , ftn mode
DirectZap	up to 8k blocks, minimum escaping, ftn mode

Also note that During batch transfers, the Last Error message field is set to "None" when starting to send or receive next file. This is to avoid the confusion caused by an error message from a previous file not being cleared.

1.4 interface

The XPR standard lays out two ways for the comm program user to specify options for the XPR. The more primitive option is for the comm program to provide a method of passing an option string to the XPR library before transferring files. The precise format and usage of this option string is left up to the XPR author; the comm program just sends it verbatim. If an environment variable is found with the same name as the XPR (i.e. there's a file in the ENV: directory called "xprzmodem"), the comm program is supposed to use this string (contents of file) to initialize the protocol options. Also, a menu option or some such should normally be provided which will allow the user to be prompted for the option string interactively.

Version 2.0 of the XPR standard created a new, more sophisticated way for the comm program user to specify XPR options. If the comm program supports it, the XPR library can give the comm program a list of option prompts, and the comm program can then let the user interactively set the various options individually, possibly even using nice gadgets and stuff.

1.5 options

T
Text Translation Mode

```
                - Controls whether or not CR/LF pairs are translated.

O
            Overwrite Mode
             - Controls what happens when a duplicate filename is
                        found.

B
            Buffer Size
             - Controls the size of the file I/O buffer.

F
            Frame Size
             - Sends an ACK after X-many bytes.

E
            Error Count
             - Determines the number of sequential errors before
                        ZModem will abort the transfer.

M
            Block Size
             - Determines the maximum block size.

A
            Auto-Activate
             - Controls whether or not ZModem will automatically
                        activate a receive.

D
            Delete After Sending
             - Controls whether or not the file is deleted after
                        it has been sent.

K
            Keep Partial Files
             - Controls whether or not partially received files
                        are kept.

R
            Receive Full Dir Path
             - Controls whether or not ZModem will use the full
                        directory path sent.

S
            Send Full Dir Path
             - Controls whether or not ZModem will send the full
                        directory path.

P
            Default Receive Path
             - Gives the default path for downloads to be.

N
            No Files Mode
             - Allows session without sending files.
```

```
  Q
                DirectZap
                 - DirectZap protocol escapes fewer characters.

  Z
                FTN mode
                 - Enables special features for Mailer operation.

  Y
                XPR2001 mode
                 - Enables XPR 2.001 extensions



                XPRzedzap Defaults
                   - Defaults for xprzedzap.library.


                XPRzmodem Defaults
                   - Defaults for xprzmodem.library.
```

1.6 Text Translation Mode

```
                              Text Translation Mode

Text Yes (TY)  - If receiving, translate CR/LF pairs or solo CR chars to
                 normal Amiga LF chars.  Ignore data past ^Z.  If sending,
                 suggests to receiver that they should receive this file in
                 text mode.

Text No (TN)   - Receive file verbatim, without changes.  If sending, suggest
                 to receiver that they receive this file verbatim, without
                 translations.

Text     (T?)  - If receiving, use sender's suggestion as to whether to do
Status           EOL translations or not.  If sending, tell receiver to use
Unknown          default mode, because we don't know either.

Text Comm (TC) - The library asks the comm program whether or not to use
                 Text mode for each file.  If the comm program doesn't
                 support the necessary xpr_finfo() call,
                 or if the call fails, this option acts like T?.  From the
                 user's point of view, what this option normally does is set
                 the Text mode to match the comm program's built-in
                 text/binary/end-of-line/translation mode, if any.

NOTE: The T option serves only as a suggestion to the receiving system when
sending files; the receiver makes the final decision as to whether to take your
advice or to force the files to be received in text or binary mode.
```

1.7 Overwrite Mode

Overwrite Mode

Overwrite Yes (OY) - If about to receive file with same name as one which already exists, delete the old file and receive the new file in its place.

Overwrite No (ON) - If about to receive file with same name as one which already exists, append ".dup" onto the name of the new file to keep them separate.

Overwrite (OR) Resume - If about to receive file with same name as one which already exists, resume receiving file data from the current end of the existing file.

Overwrite Skip (OS) - If about to receive file with same name as one which already exists, tell sender never mind, skip this file, we don't want it. Batch transfers will move on to the next file in the set, if any.

1.8 Buffer Size

Buffer Size

Buffer Size (Bnnn) - XPRZModem.library adds a layer of file I/O buffering in addition to whatever the comm program may or may not provide. This option sets the size of XPRZModem's file I/O buffer in kilobytes. The minimum value is 1 KB, for those using RAM drives or fast hard drives, or those whose comm programs already provide sufficient buffering. The maximum value is as much contiguous RAM as you have available in your Amiga.

Must be at least twice the size of M option.

If you specify more than is actually available, XPRZModem will keep decrementing the buffer size requested by 1 KB until the memory allocation works. That way, if your RAM is too fragmented to use the amount you request, XPRZModem simply uses the largest block available. Buffering is especially helpful for floppy drive users; it keeps your drive from continuously gronking and slowing things down all through the transfer. If you are a floppy drive user, you might need to set the

Frame Size

.

NOTE: Versions of VLT prior to 5.034 couldn't handle buffer sizes >= 32 KB.
If you wanted to use larger buffers before and couldn't, try again now.

1.9 Frame Size

Frame Size

Frame Size (Fnnn) - Although normally avoided, ZModem has the ability to require an ACK to be sent from the receiver to the sender every X-many data bytes. Normally you don't want to use this feature, because not waiting for ACKs is part of how ZModem works so fast. However, this feature can be very useful in conjunction with file I/O buffering on slow devices namely those floppy drives). If you set up a large I/O buffer to avoid gronking your floppy so often, you'll find that when the buffer finally *does* get around to being flushed that it can take a very long time; so long, in fact, that the delay can cause timeouts and errors. But if you set your ZModem to require the sender to wait for an ACK every buffer's-worth of data, the sender will politely wait for you to flush your buffer to the slow floppy and send it an ACK saying it's OK to continue now. This value should be set to 0 to disable ACKs (normal mode), or set it to the actual number of data bytes allowed between ACKs. For example, if you set B64 because of your floppy, you should also set F65536.

1.10 Error Count

Error Count

Error Count (Ennn) - This allows you to set the number of sequential errors which will be required to convince ZModem to abort the transfer. The normal value is 10, meaning that 10 errors must happen in a row with no valid data being transferred in order to cause an abort. This setting is provided for those using XPRZModem with a BBS, who may wish to use a relaxed setting, or those with really lousy phone lines who are desparate and patient enough to want the transfer to continue in spite of horrible performance.

1.11 Block Size

Block Size

Block Size (Mnnn) - Size of Block to transfer. Default of ZModem is 1024, minimum is 64 Bytes and the Maximum is 8192 Bytes (8K). Be careful with this option! If the uploaders blocks are bigger than the receiver because there is a older zmodem you will get errors and your cps-rate will slow down. Large blocks are useful if you have a good phoneline and a fast modem eg. 9600/14400 and higher. If you use larger blocks you will save a little bit transfer overhead and the cps-rate will get a little better. Remember, the bps-rate controls the Blocksize, this option only sets the maximum.

1.12 Auto Activate

```
                          Auto Activate

Auto-Activate Yes (AY) - If the comm program supports the ability, the library
                         will automatically go into receive mode when the start
                         of a ZModem download is detected.

Auto-Activate No (AN)  - Don't try to automatically start downloading, make the
                         user activate it.
```

1.13 Delete After Sending

```
                          Delete After Sending

Delete After Sending Yes (DY) - Delete each file after it has been sucessfully
                                sent.

Delete After Sending No (DN)  - Don't delete files after sending them.
```

1.14 Keep Partial Files

```
                                        Keep Partial Files

Keep Partial Files Yes (KY) - Keep the fragment of a file received so far if
                              file reception is aborted.  This allows you to
                              use the
                Overwrite Resume
                 option above to pick up
                              where you left off on your next attempt.

Keep Partial Files No (KN)  - Delete any partially-received file after an
                              aborted transfer.
```

1.15 Send Full Directory Path

```
                          Send Full Directory Path

Send Full Directory Path Yes (SY) - Send full filenames including directory
                                    path to receiver.

Send Full Directory Path No (SN)  - Send only simple filenames, not including
                                    directory path.
```

1.16 Receive Full Directory Path

```
                                        Receive Full Directory Path

Receive Full Directory Path Yes (RY) - Use full filename exactly as received,
                                       instead of using the P option directory
                                       path.

Receive Full Directory Path No (RN)  - Ignore received directory path (if any),
                                       use
               P
                option directory path instead.
```

1.17 Default Path

```
                                              Default Path

Default Path for (Pxxx) - Store all received files in directory "xxx" if option
Received Files
               RN
                set.  Ignored if option
               RY
                set.  "xxx" can be any
                         valid existing directory, with or without trailing
                         "/" (e.g. "Pdf0:", "PComm:hold", etc.).
```

1.18 Maximun Block size for file transfer

```
                   Maximun Block size for file transfer

            M{size}  Maximun Block size for file transfer:
               Mx = Size of Block to transfer. Default of ZModem is 1024,
                    Minimum is 64 Bytes, Maximum is 8192 Bytes (8K).
                    Be carefull with this option. If the uploaders blocks are
                    bigger than the receiver you will get errors and very
                    poor cps rates.
                    This option is normally used only in FTN mode, but may
                    be used in terminal abd BBS modem to replace the
                    XPRSZmodem.library.
                    The block size will vary when sending and will be static
                    when receiving.
                    When sending the maximum packet size will be baud rate
                    dependant, and the size is calculated with the formula
                      MAX_PACKET = (BPS_RATE * 8192 / 9600).
                    You can specify a limit for the maximum packet size with
                    the M option, but it only influences the packet size if
                    it is smaller than 8192 or if one is receiving a file
             (NOTE:  It should always be set to 8192 if one is receiving a file.
            But the option in there to limit it to less).
            The IO Buffer for reading/writing to/from the disk must be equal
            to twice the maximum packet size or the maximum packet size will
            be automatically decreased.
```

1.19 Set Link rate

```
                        Set Link rate

C{link bps}  Set bps rate of link
  C0 Buffer allocations and calculations of CPS will be based
     upon locked rate passed by the comm program.
  Cx Buffer allocations and calculations of CPS will be based
     upon link rate.
```

1.20 No files mode

```
                        No files mode

N{Y|N}  Start transfer even if no files to send:
   NY  send no files mode (DirectZap, ZedZip and ZedZap protocols)
       It is permitted to have a session without sending or
       receiving files.  This is required with some protocols in
       FTN mode so as not to generate a spurious failure after a
       mailer session.  This also changes EOF actions from sending
       CAN's to just sending ZFIN.
  NN   transfer will not take place if not files to send.
```

1.21 DirectZap mode

```
                        DirectZap mode
"
Q{Y|N}  DirectZap protocol mode:
  QY   Only ZDLE and ZDLEE are escaped.
  QN   Normal escapeing is done.
```

1.22 FTN mode

```
                        FTN Mode

Z{Y|N}  FTN mode
  ZY
   - RxTimeOut is restored to 600ms
   - transfers start with blocksize specified in M option.
   - serialbuffer is cleared before sending/recving.  In FTN
     mode the turnaround from sending to receiving (and vis-versa)
     is quite fast, clearing the buffer avoids reading echos of our
     own characters or leftovers from the previous transfer.
  ZN none of the above take place
```

1.23 XPR2001 Mode

```
                    XPR 2.001 Mode

      Y{Y|N}  XPR2001 mode
        Y - When enabled, calls to XprSetup() will return a mask with
           the  additional  bits defined in the XPR 2.001 spec related
           to  double-buffering,  etc.  xpr_update()  calls  will  be
           masked  with  a  bit  indicating  directionof  transfer  to
           support   host   program   rthat   use   dual-staus   windows.

        N - XPR2001 support diabled, required for Ncomm, Excelsior
            BBS and other hosts which do not properly handle xpr
            function and callbacks return codes.
```

1.24 XPRzedzap Defaults

```
 Default Options: xprzedzap.library

TN       No Text translation
OR       Overwrite Resume
B16      Buffer size 16KB
F0       Frame size = filelength
E30      Error count 30
SN       Do not send full directory path
RN       Do not use received full directory path
AN       Disable Auto-activate mode
DN       Do not Delete after sending
KY       Keep partial files
P""      Comm progrmas provides Path to use for received files
M8192    Maximum packet size 8K
C0       Set Link BPS Rate
NY       Alow Send if there are no files
QN       Disable DirectZap escape only CAN
ZY       Enable FTN mode
YY       Enable XPR2001 extensions
```

1.25 XPRzmodem Defaults

```
 Default options: xprzmodem.library

TC       Comm Program Sets Text translation mode
ON       Overwrite No
B16      Buffer size 16 KB
F0       Frame size = filesize
E10      Error count 10
SN       Do not Send full directory path
RN       Do not use Received full directory path
AY       Enable Auto-activate mode
DN       Do not Delete after sending
KY       Keep partial files
P""      Comm program sets Path to use for received files
```

```
M1024    Set maximum packet size 1K
C0       Set Link BPS Rate
NN       Do not Send if there are no files
QN       Disble DirectZap escape only CAN
ZN       Disable FTN mode
YY       Disable XPR2001 extensions
```

1.26 Serial Port Settings

This implementation of ZModem requires that your serial port be set to 8 data bits, no parity, 1 stop bit. This allows ZModem to send full 8-bit binary data bytes without having them munged on the way through the modem. If your comm program supports the xpr_setserial() function, XPRZModem will use it to set your serial port to 8N1 before doing a transfer, and will set your port back the way it was again after it's done. If your comm program doesn't support xpr_setserial(), you'll need to make sure it's in 8N1 mode yourself.

ZModem works well in all serial port handshaking modes; none, XON/XOFF, or 7-wire/RTS/CTS. Since any or all of those handshaking modes may be appropriate at different times, with different modems or remote systems, XPRZModem lets you set the handshaking mode and doesn't mess with it.

XON/XFF MUST be disabled when using DirectZap (option QY)

1.27 Receiving Files

Once you get the ZModem options and your serial port configuration set up properly, you're ready to actually use this thing (gasp!). Receiving files via ZModem is quite simple. First, get the file sender going by using whatever command it wants. ZModem is a batch file transfer protocol, meaning that it's capable of transferring several files in a single exchange, so the remote system may allow you to specify multiple files to be sent to you at one time. It may also allow you to use wildcard characters in the filename(s); this is all system dependant.

This may be all you have to do. If you specified option {"AY" link A} ("auto-activate" on), and your comm program supports it, XPRZModem should automatically activate at this point and start receiving your files. If you specified AN, or your comm program doesn't support auto-activation, you should now use whatever option your comm program provides to activate file reception. This will usually be a menu option or button gadget. Either way, once XPRZModem starts receiving files, it should automatically receive all of the files you specified into the proper directory as indicated by the R

and
P
options.

Make sure that you have set the ZModem options properly before starting the transfer; especially, make sure you only use
TY
if you know that all of the files being transferred in this batch are printable ASCII text files. If you use
TY
on normal binary files like .ARCs or .ZOOs, they'll be mangled beyond use.

1.28 sending

Sending files using ZModem is fairly straightforward. First, activate the file receiver with whatever command the remote system requires. You may or may not need to specify a filename or directory to the remote system; this depends on their implementation of ZModem. Once the remote system is ready to receive files, activate your comm program's ZModem send function. Your comm program will prompt you for which file(s) to send. Although ZModem is a batch protocol, your comm program may or may not allow you to specify multiple file names to be sent; also, wildcards may or may not be supported. These decisions are up to the comm program author; ZModem will handle multiple files and wildcards if the comm program allows them. Once you've specified the file name(s), the file(s) will be sent to the remote system.

If errors occur while sending the file(s), you'll probably notice a small enhancement I made to the normal ZModem error recovery procedures. Normally, file transfer protocols have to compromise between efficient data transmission on good, clean phone lines and quick error recovery on bad, noisy phone lines. If you pick a large packet size, you get high throughput on clean lines due to low packet overhead, but you have slow recovery times and large amounts of retransmitted data on noisy lines. If you've used YModem on noisy lines you've seen this problem. But, if you use small packets to reduce retransmitted data on noisy lines, you increase the amount of time the protocol spends sending packet overhead, and your throughput suffers. The solution is to vary the block size according to the experienced error rate during the transfer. That way you aren't stuck with a rigid packet length which will sometimes be the wrong size no matter what. I came up with this idea back when I first wrote the ZModem code for Opus, and cleared it for future compatibility with ZModem's designer, Chuck Forsberg, back then. Since then the basic concept has been extensively tested in the Opus BBS system, and has proven quite effective; it has also been incorporated into various other ZModem implementations over time. The actual algorithm for deciding what size packets to use when is pretty much up to the protocol author. XPRZModem uses a modified version of the Opus algorithm which prevents locking the packet size at a small value when a large one-time burst of errors occurs.

1.29 technical

Here are some notes for the "other" XPR standard users, namely the comm program authors:

Certain XPR callback functions *must* be implemented by the comm program author in order for XPRZModem to be used. If any of these functions are not supported by your comm program, XPRZModem will display an error message and abort when invoked. These required functions are:

xpr_fopen(), xpr_fclose(), xpr_fread(), xpr_fwrite(),
xpr_fseek(), xpr_sread(), xpr_swrite(), and xpr_update()

In addition, for FTN operation , the XPR v3 xpr_updstatus function is required. The library will NOT abort if your program does not have it. This function provides transfer status to the host program for EACH file sent and received.

The xpr_update() function provides many data fields for your comm program to potentially display to the user. These are the XPR_UPDATE struct elements which XPRZModem will keep updated during transfers:

xpru_protocol, xpru_filename, xpru_filesize, xpru_msg,
xpru_errormsg, xpru_blocks, xpru_blocksize, xpru_bytes,
xpru_errors, xpru_timeouts, xpru_blockcheck, xpru_expecttime,
xpru_elapsedtime, and xpru_datarate

As you can see, XPRZModem tries to provide as many status fields as possible. Although all of them are useful, the ones which are most important to ZModem users are filename, filesize, msg and/or errormsg, and bytes. Please try to provide at least these fields in your status display, plus as many of the rest as you can manage.

All callbacks are protected so we are able to call XPR callback functions in the comm program from inside the XPR library. This protects our registers from potential bugs in the comm program which might change them in unexpected ways. The prototypes in xprzmodem.h put all arguments into the registers required by the XPR spec.

Although only the XPR callback functions listed above are crucial for XPRZModem, almost all of them are used if they are provided. Although XPRZModem will function without any of the other routines, its performance or capabilities may be degraded somewhat. Specifically, this is what you give up if you choose not to supply any of these other XPR callback functions:

xpr_sflush(): Used when performing error recovery and resync when sending files. If not provided, extra timeout errors and delayed error recovery will be likely. The files will still be transferred properly, but errors will degrade overall throughput more than usual.

xpr_chkabort(): Called between sending or receiving packets. If not provided, there's no way for your comm program user to abort a transfer in progress except by trying to somehow force it to decide to give up and abort on its own, such as by turning off the modem and hoping the protocol will abort after enough timeouts (it will, eventually...).

xpr_gets(): Called to prompt the user interactively for
options
when your comm program invokes XProtocolSetup() ↩
with a null
xpr_filename field (if xpr_options() isn't available
instead). If not provided, you'll have to prompt
the user for the options string yourself, and pass this
string in xpr_filename when invoking XProtocolSetup().

xpr_setserial(): Called to obtain the current serial port
settings, and to change the
serial port
to 8N1 if it's not
already set that way. If not provided, XPRZModem will
assume all transfers are being done at 2400 bps, which
won't hurt anything, and your users will have to make sure
that the serial port is set to 8N1 themselves.

xpr_ffirst() and xpr_fnext(): If either of these routines are
missing, XPRZModem will lose the ability to send multiple
files in a batch. The xpr_filename pointer passed to
XProtocolSend() will be assumed to point to the actual full
filename of the single file to be sent in this batch. If
both of these routines are provided, XPRZModem will rely
upon them completely to obtain the names of the files to
send, and the xpr_filename pointer will not be used for any
purpose by XPRZModem except to be passed to ffirst/fnext.
This gives your comm program a way to send not just a single
filename template's worth of files in a batch, but a list of
different filenames. If, for example, you set xpr_filename
to point to the first node of a linked list of filenames
and/or templates to be sent, rather than just having it
point to a string, you can have your ffirst and fnext
routines traverse this linked list in order to determine the
next file to be sent. Or you could have xpr_filename point
to a buffer containing a list of filenames separated by
commas, and your ffirst/fnext routines could return each
filename in this list in turn. The key here is that if you
provide these two routines, you're in complete control over
the series of names fed to XProtocolSend. If you omit these
routines, XPRZModem is stuck with single-file mode. Once
again, if these two routines are provided, XPRZModem will
always call them; it makes no attempt to use the
xpr_filename pointer for anything itself. This is not
explicitly spelled out in the XPR standard, but it seems the
only reasonable way to handle batch protocols to me.
Hopefully other XPR library authors will follow this
precedent as well, so that comm program authors will be able
to count on multiple-filename batch sessions being handled
properly.

xpr_finfo(): Used to determine the filesize of files being sent,
in order to tell the receiving system how big they are.
Also used to determine the size of a file which already
exists when in
Overwrite Resume

```
          mode; XPRZModem must be able
        to get the size of the current portion of the file in order
        to be able to tell the sender where to resume sending from.
        If this routine isn't provided,
         Overwrite Resume
          mode is
        not allowed.  This routine is also used to check if
         Text mode
                        should be set to Y or N for each file when option
         TC
          is set.

    xpr_options(): If you don't supply this, users will be stuck
        with setting the library
         options
          via the semi-cryptic text
        string method (ENV: and/or xpr_gets()).  This routine and
        xpr_update() have a lot to do with the look and feel of your
        program when using XPR libraries; any skimping on these two
        routines will be painfully obvious to the user.  Conversely,
        doing a nice job on these two routines will really make your
        program shine.

    xpr_unlink(): Required by the
         DY
          and
         KN
          options, so if you don't
        supply it, those options are not allowed.
```

1.30 future

I don't want or expect this to be the last or only XPR ZModem library available. There are a lot of less-commonly-used ZModem features which have popped up over the past few years, and many people might like to see some of them made available. Although DirectZap style escaping is enabled with the QY option (everything except ZDLE and ZDLEE), 8th bit escaping to allow use of 7-bit serial channels is not on the todo list as yet.. I didn't want to add a bunch of rarely-used bells and whistles to this version of the library, because I want it to be able to serve as comprehensible example code. I just want to provide a good solid ZModem which reliably handles the majority of people's needs. Hopefully, this will serve as a foundation for future enhanced versions, while providing a safe fallback for people to come back to if that fancy new enhanced version (with neo-maxi zoomed weebies) turns out to need some more debugging.

1.31 history

```
1.0, 29 Jul 89 - Original release.

1.1,  03 Aug 89 - Fixed zsdata() to send file data packet in one swrite()
```

call instead of calling zsendline for every byte, to prevent hammering the serial.device with single-byte write requests during uploads, and to speed up effective data transmission rates.

2.0, 28 Oct 89 - Converted from Manx to Lattice C 5.04. Created prototypes and made other tweaks as required. Designed new library skeleton for Lattice code, replacing the old Manx library skeleton. Added new options TC, A, D, K, S, R, and P. Added support for xpr_options() callback routine, and generally brought things up to par with XPR Spec 2.0.

2.10, 12 Feb 91 - Fixed the following generally minor problems:

- No longer munges A6 register (this was potentially serious), and added callback glue to ensure comm program can't munge OUR registers either. Decided that the protective glue was much safer than the more elegant direct invocation used in version 2.0.
- Slightly less transmission overhead (concatenates all output into single swrites, builds output packets a bit faster).
- Considerably less receive overhead; does a lot more waiting and a lot fewer sreads, especially at high speed. WARNING: this change doesn't work with VLT version 4.846 or earlier (yes, Willy; it really was broken B-)). This change may or may not actually do you any good, depending upon how your comm program implements the xpr_sread() function.
- Fixed problems getting synchronized with some systems on uploads.
- No longer closes files twice.
- Now uses fully-reentrant sprintf() from amiga.lib; no more nasty BSS.
- A couple of obscure error messages were > 50 bytes long, causing problems with some comm program's transfer status windows, e.g. the infamouse VLT "Incredible Shrinking Status Window."
- Stabilized spastic data rate by computing elapsed time more accurately.
- Fixed sprintf() error which caused invalid filelength to be sent on uploads.
- Aligned all data for optimal performance on 68030++ CPUs (now that I have my A3000... B-)). Doesn't really make any noticeable difference, but it makes us 68030 owners feel better anyway. I'm also including a version of the library compiled for the 68020+ CPU, on the same principle.
- Now uses .DUP2 instead of .DUP.DUP, etc.
- Added config option E for number of errors which cause an abort.
- Fixed bogus IO_Torture false alarm concerning timer.device.
- Tried to fix an elusive Enforcer hit on reading location 0, but I'm not sure I really got it, since I had trouble reproducing the problem.

2.50, 15 Nov 91 - Fixed bugs and added the following features:

- Added code to support 32 bit CRC (Circular Redundancy Check). CRC-32 adds a little more protection to the data being sent and received than does CRC-16. Source for the CRC-32 additions came from the Unix version of RZ/SZ by Chuck Forsberg.

- Added code to update_rate() function in utils.c to avoid the Guru # 80000005 if you decide to adjust the system clock during an upload or download from Daylight Saving Time to Standard Time. :-)

- Proto additions using libinit.o and libent.o, and eliminating all of the assembler code was supplied by Jim Cooper of SAS. Jim also supplied the mysprintf() code to replace sprintf(). This version of XprZmodem can be compiled with the SAS version 5.10 C Compiler. I do not know how well it might compile with the Aztec compiler.

2.51, 29 Jan 92 - Rxtimeout changed from 600 to 300 for upload timeout problem by John Tillema.

2.52, 06 Mar 92 - Recompile code for 68020 library code. Non-68020 code worked fine but John Tillema was not able to test the 2.51 68020 version.

2.53, ?? ??? ?? - Special Version by Olaf 'Olsen' Barthel Author from "Term" (Don't know what he changed)

2.60, ?? ??? ?? - Don't know who made this version

2.60a,?? ??? ?? - Don't know who made this version

2.61, 3 July 93 - Rainer Hess made the following changes:

- NOT RELEASED! ONLY FOR BETATESTER.

- mysprintf() in Utils.c - returned int changed to unsigned int. SAS/C gave a Warning about this.

- In function XProtocolHostMon() (module Utils.c) declared
  static UBYTE startrcv[] = { ZPAD, ZDLE, ZHEX, "00" };
  SAS/C 6.x gave an error about this.
  Declared to:
  static UBYTE *startrcv[] = { ZPAD, ZDLE, ZHEX, '0',~'0' };

2.62, 27 Jul 93 - Rainer Hess made the following changes:

- NOT RELEASED! ONLY FOR BETATESTER.

- Now Blocksize avaiable

2.63, 30 Jul 93 - Rainer Hess made the following changes:

- NOT RELEASED! ONLY FOR BETATESTER.

- Now support locale.library to use different languages with Workbench 2.1, 3.x. On this time will be only the default english-language and a german catalog-file. Please send me

the filled up xprzmodem_catalog.ct for your language.

2.64, 3 Aug 93 - Rainer Hess made the following changes:

- NOT RELEASED! ONLY FOR BETATESTER.
- Blocksize was global declared, it's now in struct Vars.
- Bug-Fix in function update_rate(), machine crashes on little files e.g. 2 Bytes - old problem from 2.52 and before!

3.0, 13 Aug 93 - Rainer Hess made the following changes:

- It's time to make a full release...

3.1, 01 Oct 93 - Documentation update by Geoffrey Faivre-Malloy

- Conversion of documentation to Amigaguide format.

03 Oct 93 - Changes by Rainer Hess:

- ZModem runs always with the sender blocksize or uses standard-mode (M1024) if there is on one system a older zmodem.

3.2 - Code-Changes by Robert Williamson, in consultation with Yves Konigshofer.

- oversights in the credits rectified
- Added support for FTN operations and FTN Zmodem-derivative protocols
- A number of strings were not localized, corrected.
- Only wb2+ versions supported
- Removed all makefiles except 68000, 68030 and 68040 WB2 versions.
- Added callbacks.a - Protection of ALL registers used for callback functions is restored.
- XPR 3 xpr_updstatus function is enabled, permitting notification of transfer status for EACH file sent or received.
- Changed most references to KSIZE as a defined constant to use local variable v-Ksize instead.
- Compiler DEFINE ZEDZAP causes certain code sections to be replaced with xpr-friendly versions from xprzedzap source or new code inserted. These changes should be compatible with usage of the library in pure Zmodem mode.
- Defined default setups for both xprzmodem and xprzedzap optomized for each..
- partial XPR 2.001 support for dual-status windows added.
- Protocol name displayed will be one of:
  Zmodem, 1K blocks standard, up to 8K, non-ftn mode

```
        ZedZap, up to 8K Blocks based upon bps rate, ftn mode
        ZedZip, 1k blocks , ftn mode
        DirectZap, up to 8k blocks, minimum escaping, ftn mode
```

- Added localization for new options. These are NOT translated for german catalog, so that catalog has been removed from distribution.

- During batch transfers, Error message field is set to "None" when starting to send or receive next file.

3.3 - internal

3.4 - Some Protocol names were incorrect on receive, fixed.

1.32 todo

Here are some features that would be nice for XPRZModem to have sometime in the future. If there are any capable hackers out there that would like to improve upon what has already been written, feel free to do so.

- Preserve date of file being transferred.

- Investigate possibility of saving file protection bits.

- Work out ways to increase the transfer speed.

- Additional changes as time and others may suggest.

- Add TrapZap (TZA) support

- Work out ways to respond to receiver's interrupts faster.

1.33 credits

Special thanks (in alphabetical order) go to:

Olaf 'Olsen' Barthel

Jim Cooper

Geoffrey Faivre-Malloy

Rainer Hess

Rick A. Huebner

Yves Konigshofer

Willy Langeveld

Russell McOrmond

Marco Papa

William M. Perkins

John Tillema

Robert Williamson

1.34 Olaf 'Olsen' Barthel

Who knows what he changed!

1.35 Jim Cooper

Supplied the mysprintf code.

1.36 Geoffrey Faivre-Malloy

Converted XPRZModem documentation to amigaguide format.

1.37 Rainer Hess

Responsible for version 2.61 to 3.1.
Many features he claimed he added first appeared in Yves's sources.
EMail: rhess@a3tnt.adsp.sub.org

1.38 Rick A. Huebner

Wrote XPRZModem.library! Without him we wouldn't have this wonderful source code to base this on :) Rick also developed zmodem for Opus BBS on the IBM. He was also the original author of the Proteus BBS Arexx Engine for the Amiga.

1.39 Yves Konigshofer

The author of xprzedzap.library (released versions: 0.55, 0.85, 1.00, 1.5) It seems that Yves Koingshofer was not given credit for such features as dos.library usage, variable block-size, 8 K blocks etc. Perhaps these changes were done independantly to Yves's xprzedzap.library derviative or perhaps credit for his work was inadvertandly left out. Since xprzedzap.library received wide release via AmiNet, and has been in use with numerous BBS's and Term programs as well as with mailers such as JAZ, POP,

RAP, GAZEBO, PORTICUS, UMBRELLA and JAMMAIL for a number of years, I found it strange no credit was given to Yves who put so much effort into this work. Yves is also the author of Contact! BBS.

1.40 Willy Langeveld

Invented the XPR version 1.0 and had help from
Marco Papa
updating it to
version 2.0. Willy is the author of the popular telecomm program VLT.

1.41 Russell McOrmond

One of the prime movers on the XPR 3 mailing list, developed callback protection, format.a from which xprsprintf.a was copied, and the xpr_updstatus function. He is the author of the mailer development language WPL and of xprclock.library and xprfts.library. @endnode

1.42 Marco Papa

Collaborated with
Willy Langeveld
in updating the XPR specs from 1.0 to 2.0.

1.43 William M. Perkins

Spent many selfless hours of his life updating XPRZModem (2.50 & 2.52)

1.44 John Tillema

John fixed a bug in 2.51 of the library.

1.45 Robert Williamson

Developer of ver 3.2 of both xprzmodem.library and xprzedzap.library.
Keeper/Author of xprfts.library v1.1, xprslk.library v0.40.
Author of the Xpack utilities and the Shelter Mailers: Roof,Porticus,Gazebo and Umbrella and the Melody point mailers: Jaz and Rap. Keeper of the sources of ConfMail and xferq.library v1.9
email: robert@ecs.mtlnet.org