Extended XMODEM
and
Extended XMODEM/FI

Version 1.21

The Extended XMODEM specification

Release dates: 2002, 2004, 2008, 2009

Design and Implementation In-House
ADONTEC®

www.adontec.com

This document describes the Extended XMODEM extensions added to the standard XMODEM, XMODEM/CRC protocol by ADONTEC.

Technical Support

Only with consulting and support contract. Please use the contact form at http://adontec.com to receive further informations.

ADONTEC Computer Systems GmbH
HOELDERLINSTR. 32
MAULBRONN 75433
GERMANY

Foreword

Computerized data communications has already a long history and many protocols invented to secure and speed data and file transfers.

A fascinating world of bits traveling with speed of light through wire, satelites or wireless created in order to connect people and machines through distance. Very known protocols like XMODEM, YMODEM, KERMIT, ZMODEM, TCP/IP and many other invented to fill the gap and move data communication technology forward.

We proudly present to you the Extended XMODEM protocol in hope that it will serve the community and ease file transfers a bit further.

We are very eager to receive your constructive feedback.

Sincerely

Kiriakos Georgiadis

Chief Software Architect

Why invented?

XMODEM, even it is widely supported, it's rarely used, since it lacks of speed and other file transmission features that are standard today where ZMODEM rules.

ZMODEM is fine but rather hard to implement. XMODEM on the other side is easy to implement and would perform as well as ZMODEM if it would include similar and well defined features.

As computer power and communication speed was growing the throughput could get even better if the data blocks were bigger. The XMODEM-1k extension, defined in the YMODEM specification, increased the block length of XMODEM/CRC to 1024 bytes. The XMODEM-1k is driven by the sender, a fact that is not-normal to XMODEM and if the receiver does not know about it, the communication will fail.

So the requirement was to define backwards compatible extensions to an already known protocol. The *Extended XMODEM* protocol was born.

The Standard XMODEM Protocol

The standard XMODEM starts each packet with the control character SOH followed by the packet number and the inverted packet number. 128 bytes of data follow and the packet ends with the 8-bit block-check.

SOH	Block#	~Block#	DATA 0	...	DATA n	Check

The 8-bit block-check is calculated on the `DATA` part.

The protocol is driven by the receiver. The sender must wait until the receiver transmit a NAK character. The sender then starts sending packets starting with block number 1, 2, … 255, 0, 1, 2....

The last packet may include less data bytes than 128 and is therefore filled with EOF bytes (ASCII 26). Thus a file once transmitted with XMODEM „grows“ to the next multiple of 128.

XMODEM/CRC

XMODEM/CRC just replaces the NAK with the letter 'C' and the 8-bit block-check with a 16-bit CRC (*Cyclic Redundancy Check*).

The Extended XMODEM protocol

The *Extended XMODEM* protocol specification defines various buffer sizes in order to optimize the data throughput on what the data link has to offer. It uses 16 bit CRC in order to maximize detection of changes in the transmitted data. The features of this protocol can be summarized as follows:

- The protocol is based on XMODEM/CRC.
- It appends a 16-bit CRC.
- It supports packets of 128, 512, 1K, 2K, 8K, 32K or 64K bytes in length (K=1024).
- It can be easily extended to other packet length.
- It expects an 8-Bit transfer medium.
- It is backwards compatible to the standard XMODEM and XMODEM/CRC.
- It supports transmitting file information (optional feature - file info).
- Receiver informs the sender about the file name to transmit (optional feature - file request).
- Receiver can 'wake up' sender and request a specific file (optional feature - file request).
- <u>And</u> the XMODEM feature that many expected: <u>It transmits the exact file size</u> without padding the file with a number of ASCII 26 bytes!
- Using the *Extended XMODEM* protocol, the file does not „grow" since the last short packet is recognized.

Synchronization

The protocol is driven by the receiver as it is in standard XMODEM. The receiver is requesting the configuration to be used during the file transfer and is keeping track of the received packets.

The following table is showing some synchronization sequences for some variations of this protocol:

Tx		Rx
1	`'C'` `<---------------` `SOH` `---------------->`	
2	`DLE '0' 'C'` `<---------------` `DLE SOH` `---------------->`	
3	`DLE '1' 'C'` `<---------------` `DLE SOH` `---------------->`	
4	`DLE '2' 'C'` `<---------------` `DLE SOH` `---------------->`	

1. Standard XMODEM with 128 byte per block and 16-Bit CRC.
2. *Extended XMODEM* with receiver requesting 32K byte per block.
3. *Extended XMODEM* with receiver requesting 64K byte per block.
4. *Extended XMODEM* with receiver requesting 8K byte per block.

Control chars

```
DLE = one byte ASCII 16
SOH = one byte ASCII 1
EOT = one byte ASCII 4
ACK = one byte ASCII 6
NAK = one byte ASCII 21
CAN = one byte ASCII 24
'C' = the character C, ASCII 67
'1' = the character 1, ASCII 49
'2' = the character 2, ASCII 50
```

Requesting Extended XMODEM Options

The receiver selects the required *Extended XMODEM* option by transmitting three characters: DLE *option* 'C' e.g. DLE '0' 'C'in order to request block size of 32K.

If the transmitter is not aware of the extensions, it will ignore all characters up to the 'C' and start with XMODEM/CRC.

If the receiver would be sending a NAK instead, the transmitter had to do XMODEM with 8-bit checksum. So the synchronization process is rather simple and remains backwards compatible.

Block Size Options

One major enhancement in this protocol is the variable packet length (block size, buffer etc.). The following block size *options* are currently defined:

```
XM_32K_CHAR    '0'     32768 bytes per block
```
Defines a block size of 32K. Calculating the CRC is the same as in XMODEM/CRC with minor differences (see details in *CRC Calculation*).

```
XM_64K_CHAR    '1'     65536 bytes per block
```
Defines a block size of 64K. Calculating the CRC is the same as in XMODEM/CRC with minor differences (see details in *CRC Calculation*).

```
XM_8K_CHAR     '2'      8192 bytes per block
```
Defines a block size of 8K. Calculating the CRC is the same as in XMODEM/CRC with minor differences (see details in *CRC Calculation*).

```
XM_2K_CHAR     '3'      2048 bytes per block
```
Defines a block size of 2K. Calculating the CRC is the same as in XMODEM/CRC with minor differences (see details in *CRC Calculation*).

```
XM_1K_CHAR     '4'      1024 bytes per block
```
Defines a block size of 1K. Calculating the CRC is the same as in XMODEM/CRC with minor differences (see details in *CRC Calculation*).

```
XM_05K_CHAR    '5'       512 bytes per block
```
Defines a block size of 512 bytes. Calculating the CRC is the same as in XMODEM/CRC with minor differences (see details in *CRC Calculation*).

```
XM_01K_CHAR    '6'       128 bytes per block
```
The last one can be used to imitate the standard XMODEM/CRC but avoid the „grow" of the file on receiver side. Calculating and transmitting the CRC is the same as it's defined in standard XMODEM/CRC. Only the receiver will act differently on the last short packet to avoid the „grow" of the file (see the definitions for the receiver in *File Size and Last Packet* and *Synchronizing an unaware sender*). If the sender used is not aware of *Extended XMODEM* the file size will „grow". The minimum requirement to a custom sender is to transmit the last packet without adding EOF (ASCII 26) characters.

*1K=1024 bytes

Buffer Allocation

A transmitter starts sending packets for the requested length and then adds the 16 bit CRC according to the one used in XMODEM/CRC. Once the receiver received a long packet, it knows that the transmitter enabled the requested feature but it will continue keeping track of the CRC in order to catch errors and catch last short packet.

The receiver allocates an internal buffer according to the requested option increased by the size of the CRC, which is 2 bytes for the used CRC-16-CCITT.

```
Buffer = AllocMem(BufferSize + 2)
```

Calculating CRC and Last Packet

CRC calculation

The CRC calculation is the same as used in XMODEM-CRC (CRC-16-CCITT , polynomial 1021 hex) with some minor differences.

- The CRC starting value is set to hex FFFF instead of 0.
- The 1st complement of the CRC is transmitted.

The receiver's CRC, after passing the received CRC bytes also into the CRC machine, completes at the magic value hex 1D0F and not 0 as it does with standard XMODEM/CRC.

The CRC's 16 bits are transmitted as in standard XMODEM/CRC: bits 8 to 15 first (high byte) and bits 0 to 7 next (low byte).

SOH	Block#	~Block#	DATA 0	...	DATA n	CRC_HI	CRC_LO

The CRC is calculated on the `DATA` part.

Last Packet and File Size

With XMODEM, XMODEM-CRC and XMODEM-1K, it is possible for a file to "grow" up to the next multiple of 128 or 1024 bytes. With the *Extended XMODEM* the file does not „grow“ since the last short packet is recognized by the receiving algorithm used.

The receiver usually receives each packet by expecting the maximum packet length plus the CRC (2 bytes). In case of the last short packet, the receiving function will just timeout returning less bytes. If the CRC calculated matches the one received, the receiver ignores the CRC bytes in the buffer and returns the rest.

```
Count = RXPacket(Buffer, BufferSize + 2, SEC_1)
GetCRC(Buffer, Count)
ret = Count - 2     ignore the crc bytes
```

The *RXPacket* function is by default working with an inter-character timeout of 1000 ms. If no data received for this amount of time, it returns with the data received until then. If the CRC does match, then it is a complete packet, else it is a data error.

Another way to catch the last short packet without waiting to timeout is to check the amount of bytes received and compare this value with the *file size* reported by the sender.

The delay introduced by the one character timeout on the last short packet is rather meaningless. An inter-character delay of about 100 ms to 1000 ms should be sufficient for most data links. A very short value may false timeout.

*ms = 1/1000 second, 1000 ms=1 second.

Synchronizing with an unaware sender

Q: How does the receiver synchronize with an unaware sender that is sending with XMODEM/CRC instead of e.g. XMODEM/32k ?

A: The receiver must pay extra attention to the first error-free packet it receives. If the first packet received is exactly 130 (128 + 2 crc bytes) it may be: 1) the one and last short packet of a small file transmitted with the new extensions or 2) it may be the first packet of a file for which the sender does standard XMODEM/CRC and 128 byte packet size. The receiver now generates a CRC according to the standard XMODEM/CRC. If the resulting CRC is 0 we have case (2), standard XMODEM/CRC with 128 bytes packet size. If not, the receiver generates an *Extended XMODEM* CRC. If the CRC results to the magic number 1D0F we have case (1) else it's an error and it should transmit NAK to sender. See the following pseudo code:

```
Count = RXPacket(Buffer, BufferSize + 2, SEC_1)

IF (Count = 130 AND FirstPaket=FALSE) Then // Standard XMODEM/CRC ?
    crc = GetCRC(Buffer, Count, 0)  // CRC init with 0
    IF crc = 0 THEN OK = TRUE, Protocol = XMODEM_CRC, BufferSize = 128
END IF

IF NOT OK THEN   // check for new extension crc
    crc = GetCRC(Buffer, Count, FFFF)  // CRC init with 1 bits
    IF crc = 1D0F THEN OK = TRUE
END IF

IF OK THEN
    ret = Count - 2  // ignore the two crc bytes
    FirstPaket = TRUE // remember we got it
ELSE
    SendNAK
END IF
```

Speed consideration

The *Extended XMODEM* specification offers different buffer options to match better to the used speed and optimize the data throughput.

The packet size should be selected according to the speed of the data link involved. A good approach is the „1 seconds data packet“. If the used data link is serial with 115200 bps the maximum data that can be transmitted are about 11k per second. So a buffer size of 8k is fine to go. A buffer size of 32k could be acceptable too. A very large buffer size would not fail, it would possibly increase speed a bit, but any statistical information displayed by the application may not be possible, while the packet is on its way and the application could look like frozing for a few seconds.

Packet Acknowledgement Mechanism

As with the standard XMODEM, data integrity is assured by checking the block-check and transmitting ACK or NAK byte.

Timeouts

Timers are used for the several protocol states as the synchronization loop at start, the inter-character delay and the acknowledge timeout.

- WaitSYN 60s
- WaitACK 10s
- WaitOneChar 1s

The values listed here are the defaults and should be used if released to the public. Private and custom implementation may require or use other values.

Retries

Each packet transmitted may be resented, because of an error, up to 5 times. The receiver will perform various check on each packet received and NAK, if it locates any error. The retry counter used is set to 0 on each successful packet and increments by one on every retry (NAK). If the retry counter reaches the value of 6 it will initiate the abort sequence.

Graceful Abort

Aborting the file transfer may be necessary because of errors or at user request.

A user request can be identified when checking a variable that can be set by the calling application. This check is specific to the custom protocol implementation (environment, language etc.) and not part of the protocol specification.

In order to abort the protocol has to transmit the abort sequence. The abort sequence in XMODEM used to be a sequence of two CAN (ASCII 24) characters. ZMODEM used 8xCAN followed by 8xBackspaces (ASCII 8). Many variations of the abort sequence can be found in third party imlementations. *SuperCom* uses 8xCAN followed by 10xBackspaces in most of the standard protocols (XYZMODEM, Kermit) it supports.

The abort sequence is transmitted by one side only. If it is the transmitter, it clears its output buffer first and then it transmits the abort sequence. If it is the receiver, it transmits the abort sequence and keeps clearing its input buffer until no more data arrive.

If a sender is receiving the abort sequence it clears the transmit buffer and ends the protocol function. This ensures that the remote will not get unexpected data.

If a receiver is receiving the abort sequence, it continuously clears the input buffer until no more data arrive – then it ends the protocol function.

The Extended XMODEM File Information

A file transfer as offered by XMODEM was fine then but rather poor for today. A way of transmitting file information using this protocol is presented in the following.

The options described so far are mandatory and must be implemented by any software that claims to support *Extended XMODEM*. The following features are optional, but as a whole. If one claims to support the extended XMODEM file information feature (*Extended XMODEM/FI*), it must support all options described in the next section. <u>Anything less does simply not comply and such software cannot be named as such!</u>

The *File Information Packet* enables the receiver to

- use the original file name as transmitted by the sender
- use such information as file size
- time stamp the same date&time information as the original file
- check file date and file size and skip transmission if not newer
- request a specific file from sender (feature unique to this protocol)

In order to transmit file information, the sender is using the first packet with packet number 0. The sender cannot send this information if not previous told so by the receiver. This kind of information is exchanged by transmitting request and reply information using strings consisting of *name* and *value* pairs.

<u>Syntax:</u> [name=[value];]

e.g. `FILE=test.txt;` or an empty value `FILE=;`

The field names can be in any combination of uppercase or lowercase but uppercase should be preferred whenever possible in order to ease testing and debugging. Values must be presented as defined or expected by the receiver. No space (ASCII 32) allowed before and after the '='.

e.g. „`FILE=test.txt;` is the preferred way
„`File=test.txt;`‟ is acceptable too
„`FILE = test.txt;`‟ **WRONG**

Senders File Information Packet

The senders *File Information* packet may include various fields as FILE, LEN, DATE, POS and also custom fields. It's up to the receiver to use or ignore any.

The „informational“ packet is transmitted by the sender with packet number `0` and may contain a string with various fields (name + value pairs) as data.

e.g. `"`**`2034`**`;LEN=2034;FILE=test.txt;DATE=2009-10-24T20:33:45.339;",0,0`
(the characters ',' and '`"`' `are` not transmitted!)

As it is obvious from the sample, the length information is placed twice. For lazy programmers waiting only for the length information, it's placed at the beginning of this packet without the name string. <u>This file size specifier always exists</u> even if - for any reason - set to „0;“ (never expect this to be the case). This information also enables the receiver to check alternatively for the last short packet and also show statistics.

A sample of a lazy file information string having file size only: `"2034;",0,0`

The „informational" packet ends when the two zero bytes are received. The CRC will be received next.

The length of this packet is limited only by the selected buffer size.

Each field starts with the name, followed by the value and ends with a ';'. All data presented in this part should be printable data (ASCII 32 to 126) e.g.

```
    !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ\^_`
    abcdefghijklmnopqrstuvwxyz{|}~
```

The fields currently supported are presented later in this section. Custom fields may added as required.

The LEN field

The len field is prefaced with „LEN=" and defines the size of the file in bytes. Only decimal numbers allowed. If file size information is included, it enables the receiver to check alternatively for the last short packet and also show statistics.

The FILE field

The file name is prefaced with „FILE=" and can contain a path or not. The receiver can ignore the path or use it. Valid path separators are '\' and '/', but only one can be used at a time.

Example sender with file packet:

"2034;FILE=orders.txt;",0,0

The file name may be lowercase, uppercase or mixed. It does not matter to the protocol, but may be restricted by the used operating system.

The DATE field

The DATE field is prefaced with „DATE=" and may include a complete date and time or parts of it.

The Date information field, is presented in the international ISO 8601 format as YYYY-MM-DD HH:MM:SS.MS. It's valid to appear partially e.g. „Date=YYYY-MM-DD;" or „Date=YYYY-MM-DD hh:mm;" . The time representation is using the 24-hour timekeeping system and it may even include milliseconds.

ISO 8601 supported format

YYYY-MM-DD**T**hh:mm:ss

- *YYYY* defines the year [all the digits, i.e. 2012]
- *MM* defines the month [01 (January) to 12 (December)]
- *DD* defines the day [01 to 31]

The time and date can be separated with the letter 'T' or a space ' '*.

e.g. 2003-04-01**T**13:01:02

ISO 8601 does not include milliseconds* in the time field so if added it should follow the seconds starting with the period '.'.

e.g. 2003-04-01**T**13:01:02.355

---

* The following deviations from ISO 8601 introduced:

- The time and date can be separated with the letter 'T' or with a space ' '.
- Milliseconds can be added after the seconds starting the field with a period '.'.
  The following example shows this more clearly
  ```
      DATE=2003-04-01 13:01:02.355;
  ```
  Date and time separated with a space and it includes milliseconds.

The POS field

The POS field can be used by the receiver to inform an aware sender that the file transfer is to begin from a specific position in file. If this is possible the sender should include a „POS“ field in the *file information packet* it will transmit or else the receiver will ignore the resume request and will start writting from possition 0 into the file (same as „POS=0;“).

If the requested offset exists the sender should reply with the same „POS“ information. If the file is smaller than the requested offset the sender should omit or reply with „POS=0;“.

The FILESLEFT field

If more than one file is to be transmitted the sender may include additional information about the number of files and the overall file size (see FILESSIZE).

The FILESLEFT field shows the number of files remaining to transmit including the actual one. Its value starts with the total number of files and decreases by one on each following thus the last file shows „FILESLEFT=1;“.

It's expected to be included in each file of a batch, if used in the first one. If the transfer includes only one file it can be transmitted as „FILESLEFT=1;“ or omitted. If included, it enables the receiver to show statistics like: 1 of 3, 2 of 3, 3 of 3.

The FILESSIZE field

If more than one file is to be transmitted, the sender may include additional information about the overall file size and total number of files (see FILESLEFT).

The FILESSIZE field shows the total size in bytes of the remaining files included in this batch including the actual one. Its value starts with the total size of bytes and decreases by the size of the file already transmitted thus the last file shows a total file size equal to the actual file's size (=LEN).

It's expected to be included in each file of a batch, if used in the first one.

If the transfer includes only one file, it can be set to the same value as for LEN or omitted.

If this information is included, it enables the receiver to show statistics like: 1024 of 23450, 2048 of 23450, …, 23450 of 23450.

The VER field

This field informs the remote about the protocol version implemented. If used it must be „VER=1;“ for this release of the protocol.

The MFR field

This field informs the remote about who is the manufacturer of the used software or protocol.

e.g. „MFR=ADONTEC;“

The SN field

This field provides the receiver with the serial number. Must be provided if it is expected by the remote.

The USER field

This field provides login information, if required by the remote (Username).

The PASS field

This field provides login information, if required by the remote (Password).

Receiver Request Options

The receiver is requesting file information by transmitting the following data:

```
DLE buffer_option [optional options] 'C'
```

The sender triggers on DLE and filters up to the closing ']'. The rest is the standard trigger for buffer options and CRC.

So the *optional options* are set between the '[' and ']'. Care must be taken not to false trigger an unaware sender by leaving a 'C' character in this area. If the 'C' appears within this area, it should be escaped with '#' and transmitted in lowercase e.g. '#c'. If the '#' appears too it must be doubled e.g. '##'. More than one option is separated by ';'.

In this release, the fields **FILE** and **POS** are defined to be used within a receivers request and must be supported by the sender. Other fields described in the senders *Senders File Information Packet*, can also be used if expected by a custom implementation e.g. USER, PASS fields. Unaware sender will just ignore.

The FILE request field

A receiver aware of file names may request the file information by enclosing the short form of this option 'F' e.g.

`DLE4[F]C` (no spaces between the characters !)

An alternate format is using the complete name of the field.

`DLE4[FILE=;]C` Valid but meaningless use [F] instead!

or

`DLE4[`**`FILE=`**`data1.txt;`**`POS=`**`12345;]C` OK

The short form of this option ('F') can only be used alone. See below:

`DLE4[F]C`	**OK**
`DLE4[F;POS=12345;]C`	**WRONG!**
`DLE4[FPOS=12345;]C`	**WRONG!**
`DLE4[Ftest.txt;]C`	**WRONG!**

The POS request field

See the description of the POS field used in the *Senders File Information Packet*.

Requesting a specific file

The receiver may even request the sender to transmit a specific file. This is possible by adding the filename to the 'FILE' option.

Example of receiver requesting the file at C:\WORKPLACE\ORDERS.TXT

```
DLE 4 [FILE=#c:\WORKPLA#cE\ORDERS.TXT;] C
```

(where DLE = one byte ASCII 16 transmitted)

If the sender does not implement this feature, it will start with packet number 1 as in standard XMODEM. The receiver observes the starting packet number in order to handle the file information block or not.

Automated Start Of Sender

An application intending to start the sender automatically, when a receiver requests a file, has to observe the receiving buffer for the following byte sequence:

`tfex + BS + BS + BS + BS` where BS = one byte ASCII 8 (backspace)

The request is expected to start with „tfex“ and must be followed by four BS ('+' character is not really received). The four `BS` characters are used to clear the terminal window, if any used.

Once the senders protocol function executes, it has to wait for the receivers `DLE ? C` sequence.

A complete 'wake up' request from the receiver can look like the following one:

```
  tfex BS BS BS BS DLE4[FILE=#c:\WORKPLA#cE\ORDERS.TXT;]C
```

Since only the part up to the DLE is constant, the senders application can trigger up to that point. Since most protocol implementations will have the senders protocol function to expect the DLE sequence once started, this may be the preferred way to trigger.

Skip file if up-to-date

The receiver requests the file information from sender and compares the received „LEN“ and „DATE“ values with the file already on disk. If the file is up-to-date, the receiver will request the sender to **skip** this file by transmitting a CAN instead of an ACK, as reply on the senders first packet (packet number is 0). A sender receiving CAN on its *File Information* packet (first packet with packet numer 0) will just end the file transmission.

Tx		Rx
	DLE 4 [**F**] C <---------------	
	SOH|00|FF| **2034;** LEN=2034; FILE=test.txt; DATE=2009-10-24T20:33:45.339; | 0 | 0 | CRC_HI | CRC_LO ---------------->	
	CAN <---------------	

Requesting a new version of an existing file

The receiver requests the new version of a file from the sender by including the „DATE“ field and optionally the „LEN“ field with values from the file already on disk.

```
tfexBSBSBSBSDLE4[FILE=#c:\WORK\ORDERS.TXT;LEN=2536;DATE=2007-08-01T16:33:12;]C
```

If the file is up-to-date or non existent, the sender will **skip** this file by transmitting the EOT (ASCII 4) instead of its *File Information* packet and will wait for the `ACK` and then end the file transmission.

The receiver usually closes a file when `EOT` received and replies ACK. If the file is up-to-date, it should not be altered nor deleted by accident. Therefore, the receiver opens/creates the file only after it has received the first data packet (packet number=1). The result will be the untouched file in case it's up-to-date or no file at all, if the requested file does not exist on senders side.

If the „DATE“ field is provided the two dates will be checked and if the remote file is newer, the „LEN“ field will be ignored (if no other reason applies to check) and the file will be transmitted.

If the „DATE“ field is not included but „LEN“ is included, the file will be transmitted if file sizes differ.

If „DATE“ and „LEN“ fields are omitted, the file will be always transmitted if it exists.

Resume a specific file

If the receiver is aware of a broken file transfer, it may request the sender to resume from the last good position. For this, it starts the file transfer by including the „FILE“ and „POS“ fields.

```
DLE 4 [FILE=#c:\WORKPLA#cE\ORDERS.TXT;POS=2048;] C
```

Samples from the real world

1) A receiver indicating it can handle 32k blocks

```
DLE0C
```

and the sender replies with 32k data in each packet and since it should not offer file information it starts with packet number 1[1].

```
| SOH | 01 | FE | 20 | 56 | 40 | 03 | 28 | ....| CRC_HI | CRC_LO
```

2) A receiver indicating it can handle file information

```
DLE4[F]C
```

and the sender replies with packet number 0 including file information

```
| SOH | 00 | FF | 2034;FILE=sales.mdb; | 00 | 00 | CRC_HI | CRC_LO
```

3) The receiver is requesting a specific file:

```
DLE4[FILE=#c:\WORKPLA#cE\ORDERS.TXT;]C
```

and the sender replies with

```
| SOH | 00 | FF | 2034;FILE=orders.txt; | 00 | 00 | CRC_HI | CRC_LO
```

or including DATE information

```
| SOH | 00 | FF | 2034;FILE=orders.txt;DATE=2004-08-20T20:45:33; | 00 | 00
| CRC_HI | CRC_LO
```

4) The receiver does a 'wake up' call on the sender's application and is requesting a specific file as next:

```
tfex BS BS BS BS DLE4[FILE=#c:\WORKPLA#cE\ORDERS.TXT;]C
```

```
NOTE
DLE = one byte ASCII 16
BS  = one byte ASCII 8
```

The sender's reply may also include `POS, VER, DATE, FILESLEFT, FILESZISE` etc.

1 Standard XMODEM does not support file information and always starts with packet number 1.

SuperCom specifics

SuperCom was the first library used to develop this protocol.

The protocol is receiver driven and the *SuperCom* implementation offers the following constants to select the protocol option for the receiver:

PROTOCOL_EXMODEM_01K
PROTOCOL_EXMODEM_05K
PROTOCOL_EXMODEM_1K
PROTOCOL_EXMODEM_2K
PROTOCOL_EXMODEM_8K
PROTOCOL_EXMODEM_32K
PROTOCOL_EXMODEM_64K

The above constants can be used with the receiving function e.g. *RS_RXFileEx* or, in case of the ActiveX API, the property *Protocol* in order to select the preferred option.

The *SuperCom* transmitter can be called with any of the available protocol options (e.g. XMODEM, XMODEM/CRC, XMODEM/8K to XMODEM/64K). It will consider the request received from the receiver.

An unaware third party transmitter will probably start with XMODEM or XMODEM/CRC, options that are well supported by the *Extended XMODEM* receiver.

The *SuperCom* transmitter is started with *RS_TXFileEx* or, in case of the ActiveX API, the property *FileTransmit.*

The *SuperCom* receiver is started with *RS_RXFileEx* or, in case of the ActiveX API, the property *FileReceive.*

More details about using this protocol with *SuperCom* can be found in the product documentation.

New Features and Extensions

Any comments are welcomed. If you have any interesting features in mind, please, feel free to share it with us and others. From time to time we will evaluate and add new features in it. If one of the new features came from you, your name can appear next to it.

Consulting and development support

We gladly assist developers to incorporate this protocol into applications. Consulting and development services are available. You can submit your requirements through http://www.adontec.com.