YMODEM Protocol

Summary

Version: **1.01**

Last Edit: **November 21, 2014**

Author: **Andreas Isenegger**

Filename: **SummaryYMODEM_Protocol_1.01.odt**

Document Revision History

Version	Date	Author	Changes
1.01	14-Oct-2014	A. Isenegger	Corrections applied, contents added
1.00	20-May-2014	A. Isenegger	Initial version

Table 1: Document revision history

Table of Contents

1. Overview....4
1.1. History....4
1.2. Basic Block Format....4
1.3. Block 0 Data Format....5
2. Transfer Sequences....6
2.1. General Single File Transfer....6
2.2. Micro-Controller File Transfer....7
2.3. Transfer Abort Mechanism....7
2.4. Transfer Terminating Mechanism....8
3. Protocol Details....9
3.1. Checksums....9
3.2. Timeouts....9
4. Glossary....10
5. External References....11

1. Overview

The YMODEM protocol is a further development of the XMODEM protocol, and therefore very similar to it.

The XMODEM and YMODEM protocols are both half-duplex protocols.

1.1. History

The history is detailed in document [1]. Below a summary.

Term	Explanation
XMODEM	Refers to the file transfer etiquette introduced by Ward Christensen's 1977 MODEM.ASM program. The name XMODEM comes from Keith Petersen's XMODEM.ASM program, an adaptation of MODEM.ASM for Remote CP/M (RCPM) systems. It's also called the MODEM or MODEM2 protocol.
XMODEM/CRC	Is the XMODEM protocol using a 16bit CRC checksum instead of the original 1 byte XOR checksum.
XMODEM-1k	Is the XMODEM/CRC protocol using 1KiB block size.
YMODEM	Is the XMODEM/CRC protocol enhanced with optional 1KiB block size transfer possibility and batch transmission (several files in one transmission) capability.
YMODEM-g	Refers to a streaming YMODEM variation. Not described in this document since irrelevant for Bitcontrol GmbH.
True YMODEM	Is a trademarked YMODEM variant owned by Chuck Forsberg, Omen Technology.
ZMODEM	Is the successor of the XMODEM and YMODEM protocols and uses familiar technology.

Table 2: History of the XMODEM and YMODEM protocols

1.2. Basic Block Format

This block format applies to all blocks, including the first block as well as the final block. Note that the final block isn't shorter if its data field isn't full. Instead, the empty bytes in the data field shall be cleared to 0.

XOR checksum block:

Item	Header	Block Number	255 – Block Number	Data	Checksum	
Content	SOH or STX	BN	255 – BN	SOH: 128B STX: 1024B	CS	
Byte Index	0	1	2	SOH: 3 ... 130 STX: 3 … 1026	SOH: 131 STX: 1027	

Table 3: Basic Block Format of a (XOR) Checksum Mode block

16bit CRC checksum block:

Item	**Header**	**Block Number**	**255 – Block Number**	**Data**	**CRC High Byte**	**CRC Low Byte**
Content	SOH or STX	BN	255 – BN	SOH: 128B STX: 1024B	CS high byte	CS low byte
Byte Index	0	1	2	SOH: 3 ... 130 STX: 3 … 1026	SOH: 131 STX: 1027	SOH: 132 STX: 1028

Table 4: Basic Block Format of a 16bit CRC Mode block

Note: Unused bytes in the data field shall be cleared to 0 (for compatibility reasons).

1.3. Block 0 Data Format

Block 0 is also known as the *file name block* or *path name block*.

Only the data portion of the block is described here. Header, block number and CRC are described above.

The only mandatory field is the file name. The others are optional, but can't be skipped. In other words: if the UNIX access mode is provided by the sender, it also has to provide the file size and modification date. (It is unclear if they can be represented by a space character only.)

Since only the file name is a zero terminated string and the other fields are optional, these other fields are either terminated by a space character (when another field is appended) or a NULL character ('\0') (when this is the last field). When the data passed in block 0 consumes all its space, the last field may also be terminated by the data block end (no '\0' character).

File name	**File size in bytes**	**Modification date in [s]** since 01-Jan-1970 GMT	**UNIX access mode**	**File serial number**	**Zero padding**
Zero terminated ASCII string	Decimal number string (not zero terminated!)	Octal number string starting with a space character (not zero terminated!)	Octal string starting with a space character (not zero terminated!)	Octal number string starting with a space character (not zero terminated!)	Up to the end of the data block
'f' 'o' 'o' '.' 't' 'x' 't' '\0'	'3' '8' '7'	' ' '1' '2' '3' '3' '5' '4' '6' 4' '0' '3' '0'	' ' '1' '0' '0' '6' '4' '4' '4' '4'	' ' '1' '2'	0

Table 5: Block 0 Data Format

2. Transfer Sequences

2.1. General Single File Transfer

The sequence described here is a single file transfer sequence, as opposed to a batch file transfer sequence where several files are transferred.

The table below illustrates a standard sequence as used by modems.

Sender	**Receiver**
	"sx – k foo.bar<CR>" *sx: Send* *-k: Use 16bit CRC checksum* *foo.bar: File to be sent by sender*
"Sending in batch mode, etc"	
	C or NAK (periodically once per second, until sender starts transmission) *C: Use 16bit CRC checksums* *NAK: Use 8bit XOR checksums*
STX 00 FF Data[1024] CRC (CRC) *Data[1024]* contains at least the name of the file being transferred	
	ACK C or NAK *C: Use 16bit CRC checksums* *NAK: Use 8bit XOR checksums*
STX 01 FE Data[1024] CRC (CRC)	
	ACK
STX 02 FD Data[1024] CRC (CRC)	
	NAK
STX 02 FD Data[1024] CRC (CRC)	
	ACK
SOH 03 FC Data[128] CRC (CRC)	
	ACK
EOT	
	ACK C or NAK (periodically once per second, until sender starts transmission) *C: Use 16bit CRC checksums* *NAK: Use 8bit XOR checksums*
SOH 00 FF NUL[128] CRC (CRC)	
	ACK

Table 6: Block Sequence used by modems

2.2. Micro-Controller File Transfer

The table below illustrates a sequence as used between terminal programs and micro-controller boards used by Bitcontrol GmbH.

Sender	**Receiver**
	C (periodically once per second, until sender starts transmission)
STX 00 FF Data[1024] CRC CRC *Data[1024]* contains the name of the file being transferred, its size in bytes and other data, which is irrelevant in this context	
	ACK C
STX 01 FE Data[1024] CRC CRC	
	ACK
STX 02 FD Data[1024] CRC CRC	
	NAK
STX 02 FD Data[1024] CRC CRC	
	ACK
STX 03 FC Data[1024] CRC CRC	
	ACK
EOT	
	ACK C (periodically once per second, until sender starts transmission)
STX 00 FF NUL[1024] CRC (CRC)	
	ACK

Table 7: Block sequence used by micro-controller boards

2.3. Transfer Abort Mechanism

A transfer can be aborted by either the sender or the receiver.

- If the sender wants to abort the transfer, it sends two consecutive CAN control bytes instead of the next SOH or STX control byte. (todo: To be verified using Tera Term.)
- If the receiver wants to abort the transfer, it sends two consecutive CAN control bytes instead of an ACK or NAK control byte after receiving a block.

2.4. Transfer Terminating Mechanism

A transfer is completed by the sender by sending an EOT control byte. It is repeated up to 10 times with 1 second delay in between, until the sender receives an ACK from the receiver.

3. Protocol Details

3.1. Checksums

The protocol supports two different checksum modes:

- 1byte/8bit XOR checksum, called C*hecksum Mode*
- 2byte/16bit CRC checksum, called *CRC Mode*
  The checksum type is called *CCITT* or *X25*. Its generator polynomial is `X^16 + X^12 + X^5 + X^0 = 0x1021`, and its start value is 0.

The checksum comprises the payload bytes of a block:

- Bytes 3 to including 130 in a 128byte large block, counting starts at 0
- Bytes 3 to including 1026 in a 1024 large block, counting starts at 0

By including the checksum byte(s) in the checksum calculation, the resulting checksum value is zero when no checksum error has occurred, and non-zero otherwise. This is true for both checksum modes.

3.2. Timeouts

The timeout between 2 bytes is 1 second.

4. Glossary

0x<number> Number in hexadecimal format; example: 0x14 equals 20 in decimal format

Protocol Control Bytes:

ACK	0x01: Acknowledge transmission
C	0x43 (character 'C'): Receiver requests CRC16 checksum
CAN	0x18: Cancel (abort) transfer
CRC	Cyclic Redundancy Code
EOT	0x04: End Of Transfer
NAK	0x15: **N**ot **A**c**k**nowledge transmission (request retransmission), or request for 8bit XOR checksum
SOH	0x01: **S**tart **O**f **H**eader of a standard block, block size: 128B
STX	0x02: **St**art of header of an e**x**tended block, block size: 1024B

5. External References

- [1] XMODEM and YMODEM protocol specifications written by Chuck Forsberg, local file, name: *XMODEM_YMODEM_ProtocolSpec.txt*
- [2] Terminal Emulation Program written by T. Teranishi, now owned by the Tera Term Project: *Tera Term, version 4.78*