

Contents - CAN-Do Byte-pipe File-transfer Protocol (CDFTP)

Front Matter

Contents	Sheet 1
Notes	Sheet 2

Flowchart: Write File

Write File 1of2	Sheet 3
Write File 2of2	Sheet 4

Flowchart: Read File

Read File 1of3	Sheet 5
Read File 2of3	Sheet 6
Read File 3of3	Sheet 7

Message Formats:

Normal Messages	Sheet 8
Extended Headers	Sheet 9

Examples:

Successful Transfers	Sheet 10
Aborted Transfers	Sheet 11

Appendices

A - CRC Algorithm Examples	Sheet 12
----------------------------	----------

Back Matter

Change History	Sheet 13
----------------	----------



PROTOCOL Notes - CAN-Do Byte-pipe File-transfer Protocol

transferring files over byte-pipe

There will be a number of Byte-Pipe devices on our satellites. In order to simplify implementation of communication with many different byte-pipe devices we are specifying a file transfer protocol which is to be implemented by all byte-pipe devices. This enables the IHU developers to implement a single file transfer protocol which will work for all devices onboard the spacecraft.

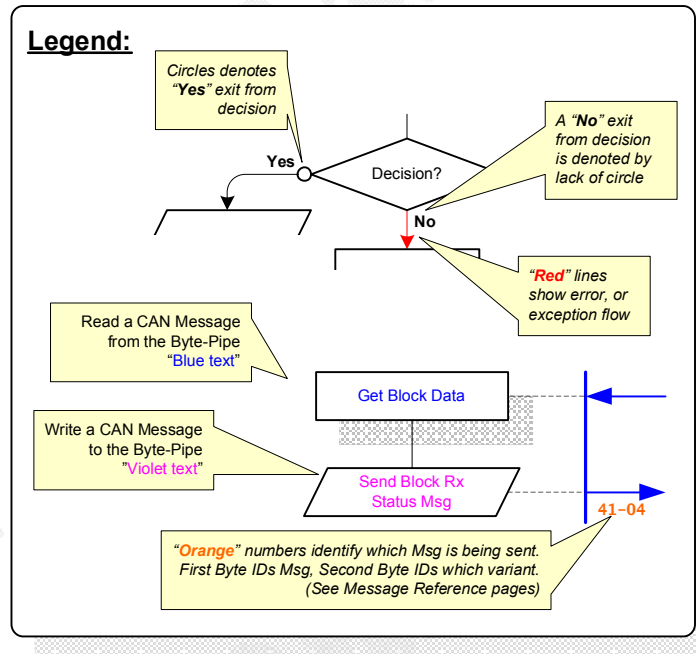
Key Features

Based loosely on the FLASH programming protocol used for the ATMEL T89C51CC01 part onboard the CAN-Do Widget.

Supports requestor defined block size.

Added:

- **CRC16 wrapped blocks**
facilitating checking of blocks, enabling block retransmission.
- **Communication of overall block count**
Enables verification that all blocks have been received.
- **Communication of block numbers**
Enables verification of block receipt order.
- **Support for Transfer Aborts**
Allows aborts to occur without having to re-power the device to regain synchronization with the device
- **Block retry count at discretion of Controller, not Byte-Pipe Device**
- **Protocol control packets CRC'd**
Affords easy checking of damaged packets or unexpected packets



Terms used in this document

Message

8-Bytes of data sent over the CAN Bus as the payload of a PIPE-WRITE or PIPE-READ CAN Message.

Block

A block consists of one or more messages (the number of which is set during the initial transfer request). This number is fixed for the entire transfer except for the final block which can be fewer messages.

Only blocks are wrapped by a CRC16 check values.

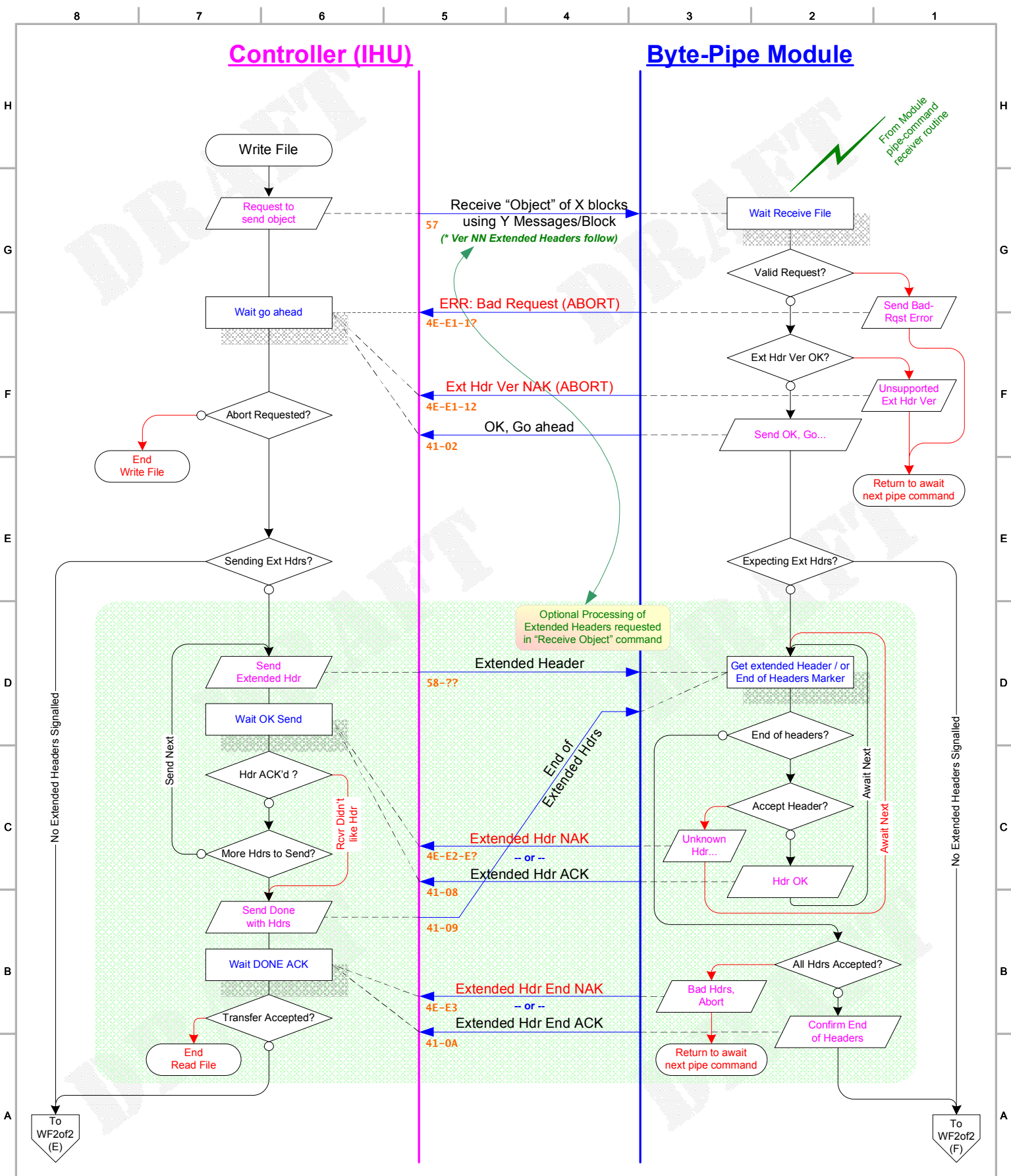
Block Header

A message describing the upcoming block. Carries, at least, the block sequence number and the checksum of the block.

Extended Header

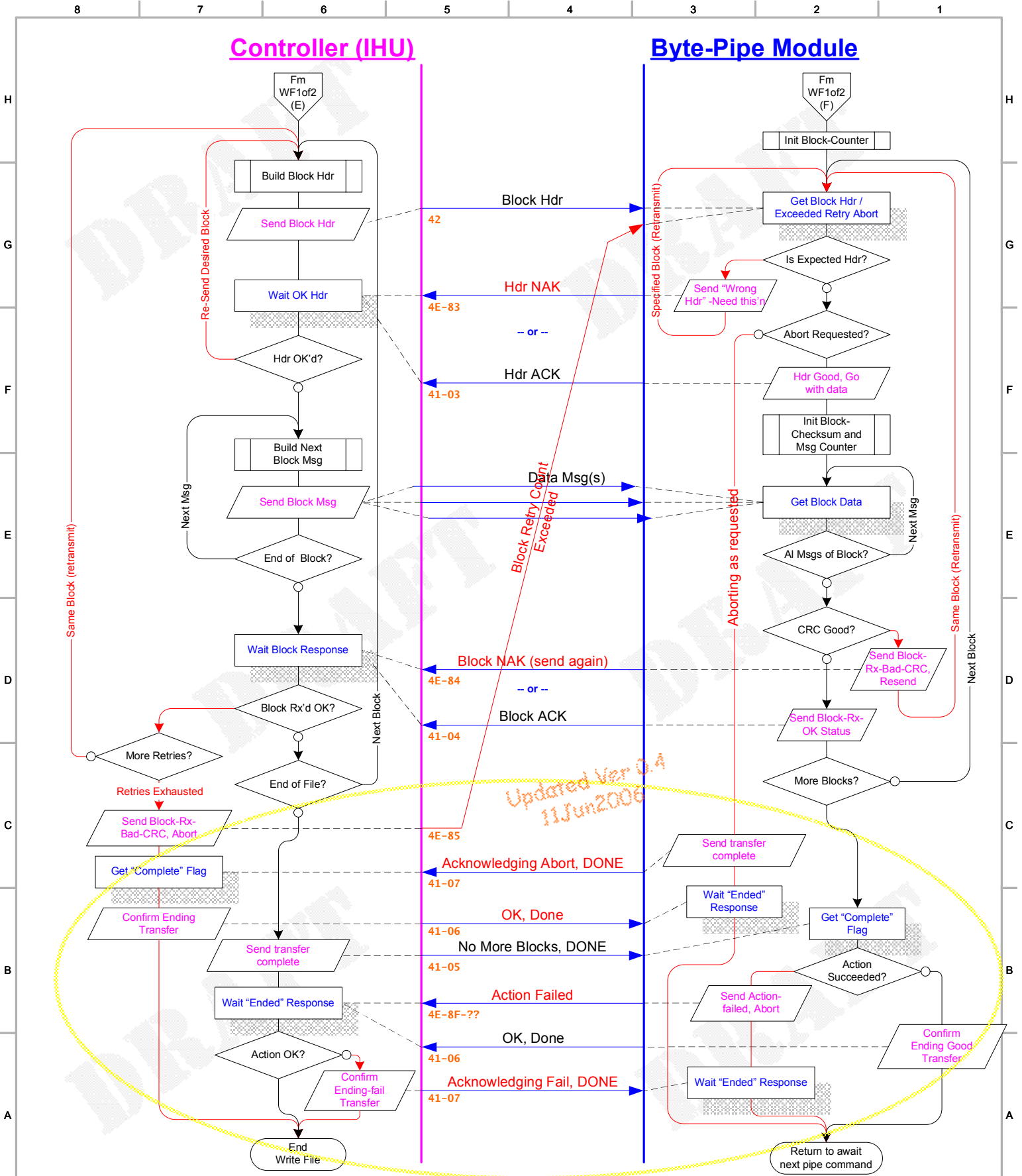
A message identifying additional information for a given transfer. Extended Headers add to the fundamental capability of the protocol and provide a means by which we can extend the protocol in the future.





Controller (IHU)

Byte-Pipe Module



AMSAT-NA

DRAFT

Drawn By: Stephen Moraco, KC0FTQ

CAN-Do - Protocol:
File Transfer via Byte-Pipe

Sheet: Write File 2of2

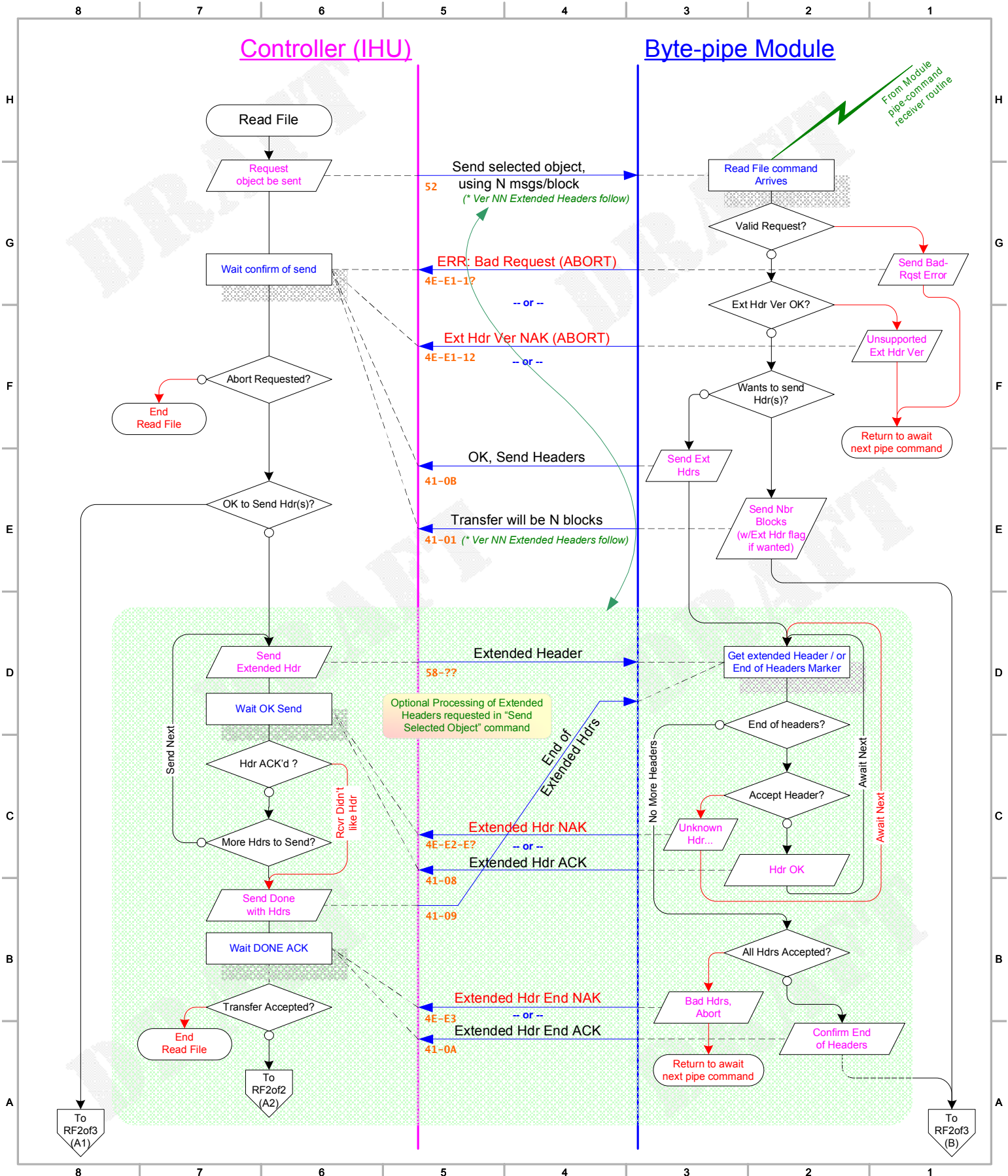
11 Jun 2006 22:00 MDT

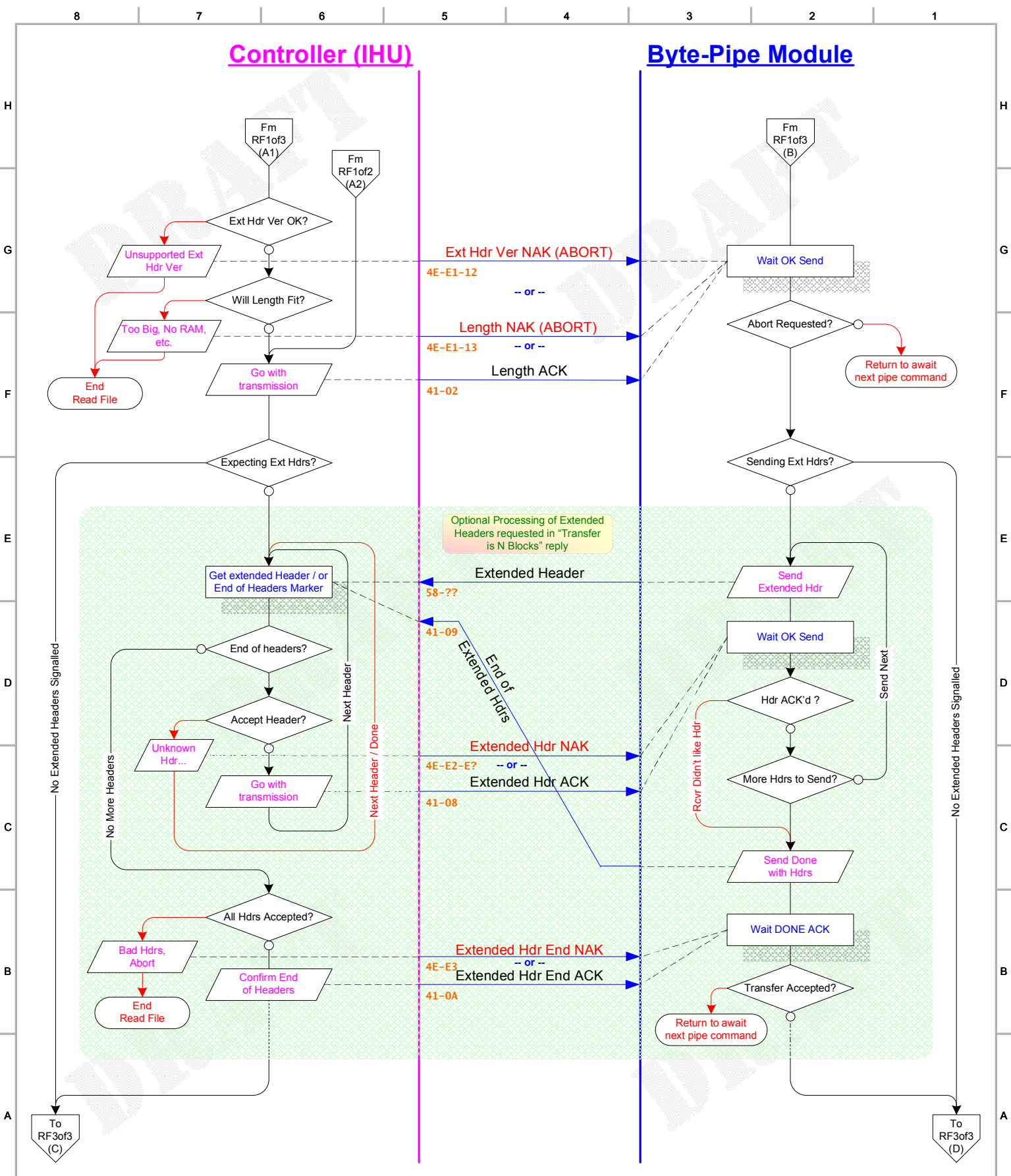
Sheet 4 of 13

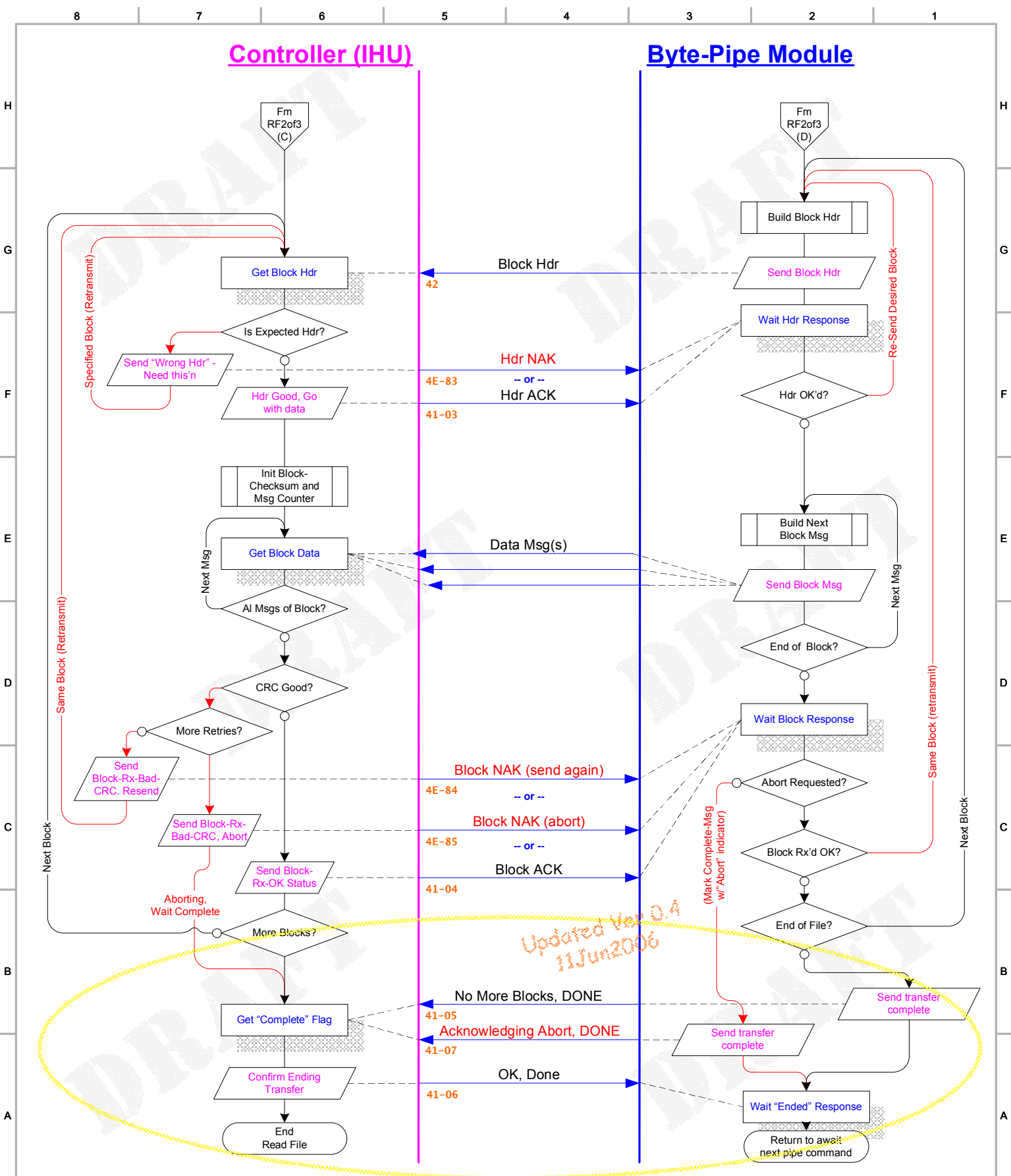
0.4

Controller (IHU)

Byte-pipe Module







Message Reference - CAN-Do Byte-pipe File-transfer Protocol

Protocol version 0 - (first formal version)

This page along with the [Extended Headers Reference](#) page comprise the complete set of messages used in this protocol. These messages are all payload definitions for **Pipe-Write** or **Pipe-Read** "CAN-Do! Widget Protocol" Messages. All of these payloads are 8-bytes long. The bytes are numbered 0 (zero) to 7. The bits within each byte are number 0 to 7 as well with the most significant bit being bit number 7. Bytes/bits marked (Reserved) have no meaning and should be set to zeros.

0x57 Write File

[to Byte-pipe Module from Controller (IHU)]

Message Bits	Purpose
0.7-0	Command (0x57)
1.7-4	Protocol Version [0-15]
1.3	{Reserved}
1.2-0	Version [1-7] Extended Headers follow (0 means no hdrs)
2.7-0	Messages per Block (Block Size)
3.7-0	[Module Defined] Object Description in Module Terms
4.7-0	MS-Byte - Number of Blocks [> 0]
5.7-0	mid-byte
6.7-0	LS-Byte
7.7-0	Checksum Byte

NOTE: Byte 3 = 0xFF specifies **Raw Mode**. Raw Mode writes allow the Controller (IHU) to write to any memory region directly. Extended headers identify which memory region (RAM, FLASH, EDAC, etc.) is to be written. Additionally, an extended header can be specified which specifies that after the memory is written, the code should be executed starting from a specific address. In the case of 0xFF the extended header version field **_MUST_** be non-zero. **Memory regions supported, the exact address range of each, and which regions support "execute from" are all Module Defined!**

0x52 Read File

[from Byte-pipe Module to Controller (IHU)]

Message Bits	Purpose
0.7-0	Command (0x52)
1.7-4	Protocol Version [0-15]
1.3	{Reserved}
1.2-0	Version [1-7] Extended Headers follow (0 means no hdrs)
2.7-0	Messages per Block (Block Size)
3.7-0	[Module Defined] Object Description in Module Terms
4.7-0	{Reserved}
5.7-0	{Reserved}
6.7-0	{Reserved}
7.7-0	Checksum Byte

NOTE: Byte 3 = 0xFF specifies **Raw Mode**. Raw Mode reads allow the Controller (IHU) to read any memory region directly. Extended headers identify which memory region (RAM, FLASH, EDAC, etc.) is to be read and how many block of memory from this address are to be returned. In the case of 0xFF the extended header version field **_MUST_** be non-zero. **Memory regions supported and the exact address range of each is Module Defined!**

0x42 Block Header

[Write file: from Controller (IHU) to Byte-pipe Module]
[Read File: from Byte-pipe Module to Controller (IHU)]

Message Bits	Purpose
0.7-0	Command (0x42)
1.7-0	MS-Byte - Block CRC (CCITT CRC16)
2.7-0	LS-Byte
3.7-0	Non-last block: 0, Last block: Number messages if short
4.7-0	MS-Byte - Block Number [1-N]
5.7-0	mid-byte
6.7-0	LS-Byte
7.7-0	Checksum Byte

NOTE: Byte 3 **_MUST_** be zero for all blocks except the final block. Only when the final block is short should this field be set to the actual number of messages in the final block.

0x58 Extended Header

[either direction based on location in protocol]

Message Bits	Purpose
0.7-0	Command (0x58)
1.7-5	Header ID Version [1-7, 0 Not Used]
1.4-0	Header ID [0-31]
2.7-0	{Reserved}
3.7-0	{Reserved}
4.7-0	{Reserved}
5.7-0	{Reserved}
6.7-0	{Reserved}
7.7-0	Checksum Byte

All Extended Headers occupy this message shape.
Each header-id has a different definition for bytes 2 - 6.
[See [Extended Header Ref.](#) page for details]

0x4E Object NAK

[either direction based on location in protocol]

Message Bits	Purpose
0.7-0	Command (0x4E)
1.7-0	Parameter (what's being negatively acknowledged)
	0x8? READ-FILE/WRITE-FILE NAK, general error, why:
0x83	- BLOCK HDR NAK, wrong seq # (need # [Aux Parm])
0x84	- BLOCK DATA NAK, resend this block
0x85	- BLOCK DATA NAK, (retry count exceeded) abort
0x8F	- Module Error (detail [MsEc byte-3])
	0xE? READ-FILE/WRITE-FILE NAK, bad request, why:
0xE1	- Unknown/Invalid Transfer ([Descriptor])
0xE2	- Unknown/Invalid Extended Header ([Descriptor])
0xE3	- Invalid Transfer spec'd by ExtHdrs, abort
	Descriptor (error detail)
2.7-0	0x1? WHY: Transfer error reasons:
0x11	- Transfer Protocol Version not supported
0x12	- Extended Header Version not supported
0x13	- FILE-LENGTH NAK, can't handle, abort
0x14	- Requested Object not known by Module
0x15	- Block size not supported
	0xE? WHY: Extended Header reasons:
0xE8	- Invalid Memory Region (unknown to this Module)
0xE9	- Addr out of range (memory region not that big)
0xEA	- Address not executable
	Module-specific Error-code (when Parameter=0x8F)
0x01	{Reserved AUX} (Alt: Aux Parameter Field)
5.7-0	{Reserved AUX} "
6.7-0	{Reserved AUX} "
7.7-0	Checksum Byte

Aux-Parameter is Desired Block when parameter=0x83: Block Hdr NAK

4.7-0	MS-Byte - Block number to resend [1-N]
5.7-0	mid-byte
6.7-0	LS-Byte

0x41 Object ACK

[either direction based on location in protocol]

Message Bits	Purpose
0.7-0	Command (0x41)
1.7-0	Parameter (what's being acknowledged)
0x01	- READ-FILE ACK, here's length ([Aux Parm])
0x02	- READ/WRITE-FILE ACK, length good, send blocks
0x03	- BLOCK HDR ACK, send block data
0x04	- BLOCK DATA ACK, send next block
0x05	- BLOCK ACK, ACK, no more blocks XFER is COMPLETE
0x06	- TRANSFER COMPLETE ACK
0x07	- ABORT COMPLETE ACK, TRANSFER is ENDED
0x08	- ExtHdr ACK, Extended Header understood
0x09	- ExtHdr ACK, ACK no more extended Headers
0x0A	- ExtHdr Handling is COMPLETE
0x0B	- READ-FILE ACK, Send ExtHdrs
	{Reserved}
2.7-0	{Reserved AUX} (Alt: Aux Parameter Field)
3.7-0	{Reserved AUX} "
4.7-0	{Reserved AUX} "
5.7-0	{Reserved AUX} "
6.7-0	{Reserved AUX} "
7.7-0	Checksum Byte

Aux-Parameter is Length in blocks when parameter=0x01: READ-FILE ACK

3.7-0	MS-Byte - Number of Blocks [> 0]
4.7-0	mid-byte
5.7-0	LS-Byte
6.7-3	{Reserved}
6.2-0	Version [1-7] Extended Headers follow (0 means no hdrs)



Extended Header Ref. - CAN-Do Byte-pipe File-transfer Protocol

Headers version 1 - (first formal version)

This page along with the **Message Reference** page comprise the complete set of messages used in this protocol. These messages are all payload definitions for **Pipe-Write** or **Pipe-Read** "CAN-Do! Widget Protocol" Messages. All of these payloads are 8-bytes long. The bytes are numbered 0 (zero) to 7. The bits within each byte are number 0 to 7 as well with the most significant bit being bit number 7. Bytes/bits marked (Reserved) have no meaning and should be set to zeros.

0x58-0x21 Transfer Start Address for Raw-Mode Read/Write

Message Bits	Purpose
0.7-0	Command (0x58)
1.7-5	Header ID Version = 0x1
1.4-0	Header ID = 0x1
2.7-0	MS-Byte - Transfer-start Address
3.7-0	mid-byte
4.7-0	mid-byte
4.7-0	LS-Byte
6.7-0	Memory Region [Values defined by specific Module]
7.7-0	Checksum Byte

0x58-0x26 AZ-Star image dimensions for Read-File

Message Bits	Purpose
0.7-0	Command (0x58)
1.7-5	Header ID Version = 0x1
1.4-0	Header ID = 0x6
2.7-0	MS-Byte - X-Axis Dimension
3.7-0	LS-Byte
4.7-0	MS-Byte - Y-Axis Dimension
4.7-0	LS-Byte
6.7-0	{Reserved}
7.7-0	Checksum Byte

0x58-0x22 Count of Blocks for Raw-Mode Read-File

Message Bits	Purpose
0.7-0	Command (0x58)
1.7-5	Header ID Version = 0x1
1.4-0	Header ID = 0x2
2.7-0	MS-Byte - Block Count [1-N]
3.7-0	mid-byte
4.7-0	LS-Byte
4.7-0	{Reserved}
6.7-0	Memory Region [Values defined by specific Module]
7.7-0	Checksum Byte

0x58-0x23 Execute Address for Raw-Mode Write-File

Message Bits	Purpose
0.7-0	Command (0x58)
1.7-5	Header ID Version = 0x1
1.4-0	Header ID = 0x3
2.7-0	MS-Byte - Execution Address
3.7-0	mid-byte
4.7-0	mid-byte
4.7-0	LS-Byte
6.7-0	{Reserved}
7.7-0	Checksum Byte

0x58-0x24 File Basename for Read-File

Message Bits	Purpose
0.7-0	Command (0x58)
1.7-5	Header ID Version = 0x1
1.4-0	Header ID = 0x4
2.7-0	StrByte[0]
3.7-0	StrByte[1]
4.7-0	StrByte[2]
4.7-0	StrByte[3]
6.7-0	StrByte[4]
7.7-0	Checksum Byte

0x58-0x25 Date/Time for ...

Message Bits	Purpose
TBD	



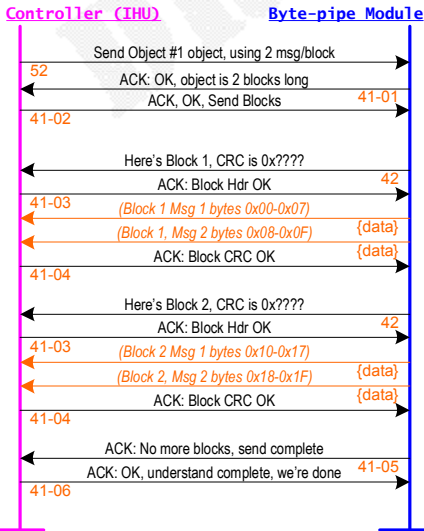
Transfer Examples - CAN-Do Byte-pipe File-transfer Protocol

(Successful normal transfers and successful transfers requiring extended headers)

NOTE: Data blocks are shown in Orange Text and Extended headers are shown in Green Text.

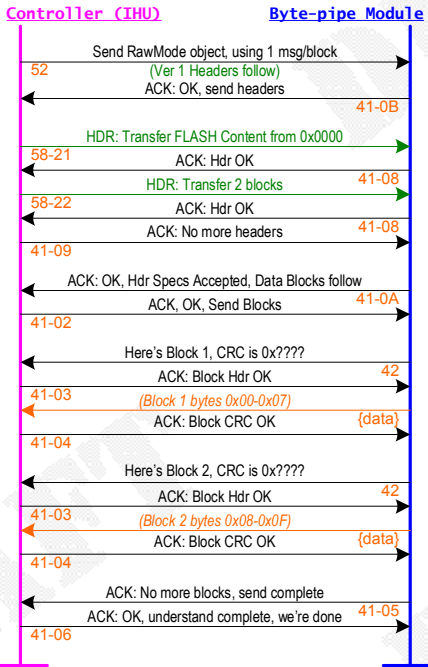
Ex1: Normal Read

Read Module-Object #1 using two messages per block.



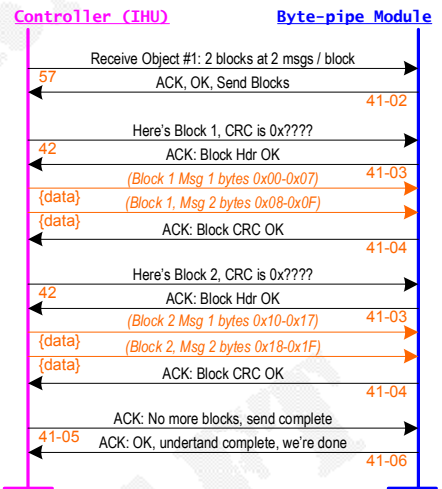
Ex2: RAW Read

Read the 16 bytes of FLASH at address 0x0000 from Byte-pipe Module using 1 message per block.



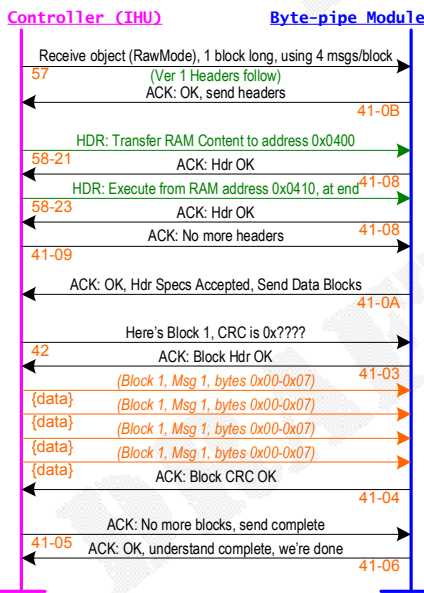
Ex3: Normal Write

Write Module-Object #1 using two messages per block, it's 2 blocks long.



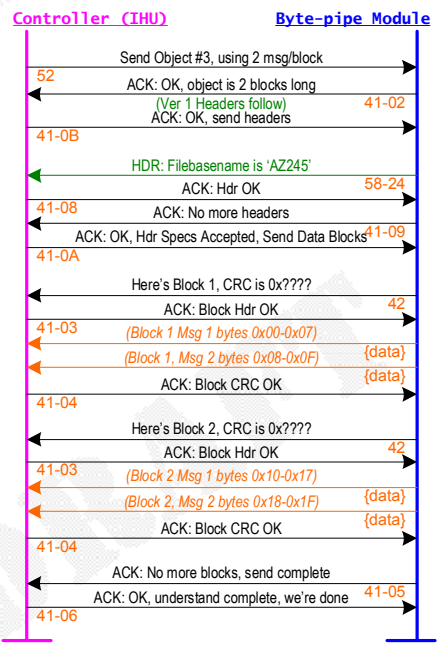
Ex4: RAW Write w/Exec

Write 32 bytes into RAM at address 0x0400 to Byte-pipe Module using 4 message per block and execute from address 0x0410 after write is complete.



Ex5: Read w/Filename

Read Module-Object #3 using two messages per block. Module specifies 5-bytes of the filename for the controller to use when storing the file.



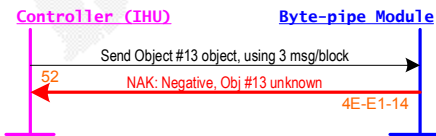
Transfer Examples - CAN-Do Byte-pipe File-transfer Protocol

(Various transfer Aborts, Corrections)

NOTE: Data blocks are shown in **Orange Text**, Extended headers are shown in **Green Text**, and Exceptions are shown in **Red Text**.

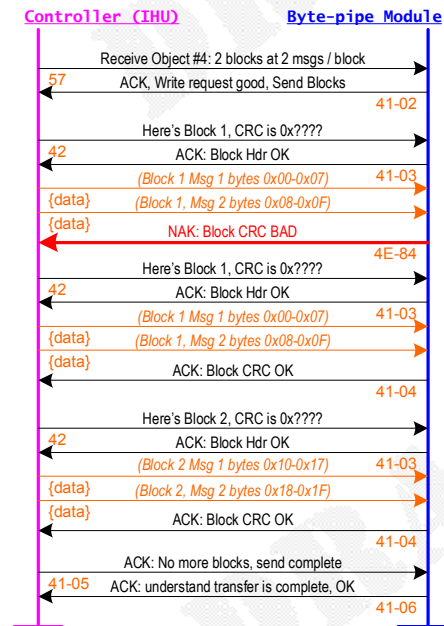
Abt1: Bad Request

Read Module-Object #13 using three messages per block but Module doesn't know of Obj#13.



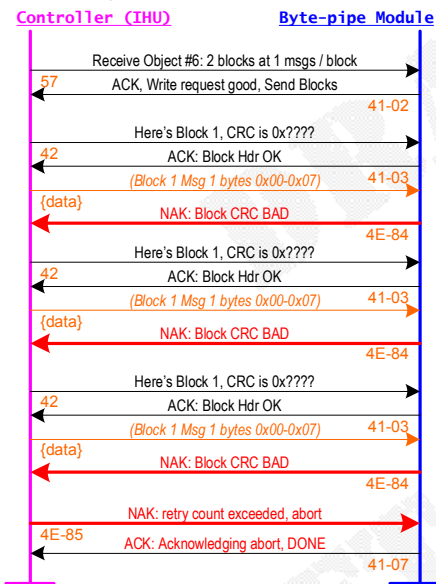
Cor1: Block bad, Resend

Write Module-Object #4 using two messages per block, it's two blocks long. First block sends badly, is resent. Rest of transmission succeeds.



Abt2: Exceeded retry count

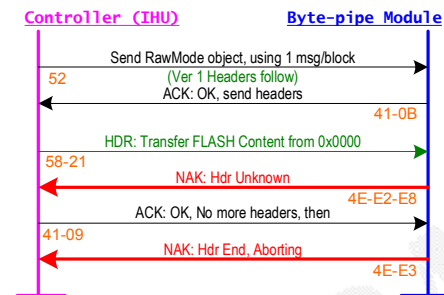
Write Module-Object #6 using one message per block, it's two blocks long. First block sends badly, is resent until retry count is exceeded, transmission fails.



NOTE: in this example the Controller is configured to allow only three retries when resending data blocks. In both file reads and file writes this decision is up to the Controller.

Abt3: Bad Ext Header

Read FLASH starting from 0x0000 using one message per block but Module doesn't know of "FLASH".



Appendix A - CAN-Do Byte-pipe File-transfer Protocol

(CRC Algorithm Examples in various languages)

CRC16: Data Blocks within File being transferred

LANGUAGE: ANSI C

```
/*
 * Use the good CCITT implementation of the CRC16 algorithm.
 * For a new CRC16, the crc argument should initially be set to 0xFFFF
 */
#define CRC16_POLY    0x1021        // CRC16-CCITT mask

/*
 * CRC16 implementation according to CCITT standards
 */
UInt16
crc16_ccitt(UInt16 crc, const void *buf, Int32 len, boolean augment)
{
    register UInt32    counter;
    register UInt8    yoda;
    register UInt8    i;
    register UInt8    v;
    register UInt8    xor_flag;

    for (counter = 0; counter < (UInt32)len; counter++)
    {
        yoda = *(UInt8 *)buf++;

        v = 0x80; // Align test bit with leftmost bit of the message byte.

        for (i = 0; i < 8; i++)
        {
            if (crc & 0x8000)
                xor_flag = 1;
            else
                xor_flag = 0;
            crc = crc << 1;

            if (yoda & v)
            {
                // Append next bit of message to end of CRC if it is not zero.
                // The zero bit placed there by the shift above need not be
                // changed if the next bit of the message is zero.
                crc = crc + 1;
            }

            if (xor_flag)
                crc = crc ^ CRC16_POLY;

            // Align test bit with next bit of the message byte.
            v = v >> 1;
        }
    }

    if (augment)
        return augment_crc16(crc);
    else
        return crc;
}
```

```
/*
 * This function should be called when all data has been
 * processed to calculate the CRC value. This will augment
 * the CRC to make sure it is correct.
 */
UInt16
augment_crc16(UInt16 crc)
{
    register UInt8    i;
    register UInt8    xor_flag;

    for (i = 0; i < 16; i++)
    {
        if (crc & 0x8000)
            xor_flag = 1;
        else
            xor_flag = 0;
        crc = crc << 1;

        if (xor_flag)
            crc = crc ^ CRC16_POLY;
    }

    return crc;
}
```

Custom CRC8: Protocol Command/Status Packets

LANGUAGE: ANSI C

```
/*
 * Calculate an 8bit checksum value. The parameter 'chksum' is the starting
 * checksum value. For a new checksum, chksum should be initialized to 0x5E.
 */
UInt8
checksum(UInt8 chksum, UInt8 *data, UInt32 len)
{
    UInt32    i = 0;

    while (i < len)
    {
        chksum = chksum ^ (((*data << i) >> 8) + ((*data << i) % 256));
        data++;
        i++;
    }

    return chksum;
}
```

LANGUAGE: Python

```
def csum(inp):
    ret=0x5E
    for x in range(7):
        ret=ret^cl8s(inp[x],x)
    return ret
```

Cl8s() is a cyclic 8 bit left shift, a simple way to do so is as follows:

```
def cl8s(num, shift):
    return (num<<shift)/
    256+ (num<<shift)%256
```



Change History - CAN-Do Byte-pipe File-transfer Protocol

(Record of changes with each document release)

V0.4 - 11Jun2006

- Adjustments based on feedback from initial AZ-Star implementation
- Last block can now be short (see 0x42 Block Hdr)
- 0x58-0x21 Ext. Hdr. Now applicable to both raw-read/raw-write
- 0x58-0x22 Ext. Hdr. now only applicable to raw-read
- Initial Bad-Request feedback on read/write flows corrected (showed non-error return)
- NEW late write-abort sequence added for Module specific aborts (*See bottom of Write-file 2of2*)
- NEW read-terminating abort added (*See bottom of Read-file 3of3*)
- NEW Extended header added for AZ-Star use
- Cleaned-up ACK/NAK code descriptions
- Added block-ids, ack/nak codes to examples
- Added example aborts and corrective sequences
- Added Appendix A: CRC Algorithms

V0.3 - 17Apr2006

- First version considered ready for implementation
- Number-Blocks field shortened to 24bits
- All Command/Header/Status blocks now contain checksum as final byte
- Final command codes defined (missing read/write codes added)
- Command codes kept outside of AZ Star Camera Codes (00-3f)

V0.2 - 19Feb2006

- Changes rolled-in from all reviewers
- Added Extended Headers
- Extensive rework of flow-charts

V0.1 - 06Feb2006

- First Draft for Review