1. Serial Protocol v4 2
1.1 Commands Reference 8
1.1.1 Application Command - Get Data 9
1.1.2 Application Command - Get Status 10
1.1.3 Application Command - Send 11
1.1.4 Authenticate - step 1 12
1.1.5 Authenticate - step 2 13
1.1.6 Get Configuration Value 14
1.1.7 Get string length 16
1.1.8 Oscilloscope Control 17
1.1.9 Oscilloscope Read 19
1.1.10 Pipe Control 20
1.1.11 Pipe Information 21
1.1.12 Read Memory 23
1.1.13 Read Memory with Base Address 24
1.1.14 Recorder Control 25
1.1.15 Recorder Status 27
1.1.16 TSA Control 29
1.1.17 User Resource Read/Write/IOCTL 32
1.1.18 Write Memory 36

Serial Protocol v4

Author	Michal Hanak
Description	Design protocol supporting new FreeMASTER 3.0 features
Status	DRAFT
Reviewers	Petr Gargulak, Vilem Zavodny

Introduction

FreeMASTER 3.0 defines and implements new serial protocol to support all features required in Low-Level Protocol Requirements and MCU Driver Requirements. Formally, the protocol version number is 4 (this is because v3 was used with FreeMASTER 1.2 already).

Backward Compatibility

The protocol is not required to be backward compatible with previous version. The PC-side tools will support both protocols so from a customer perspective, everything remains compatible. The new MCU driver included in MCUXpresso SDK will only support new protocol.

Old versions FYI

- The old protocol version (v3) is documented in the CHM help file: mcbcom.chm
- Old version of the MCU driver is documented here: FMSTRSCIDRVUG.pdf

What is not changed?

Serial protocol physical transport does not change

- UART physical transport still uses a principle of SOB (0x2b) byte which restarts the frame. When 0x2b appears in the payload, it is duplicated. So when a single SOB appears, followed by a different byte, it causes the receiver state machine to reset.
    - Duplicating of SOB character is aplied on all bytes except the real start character followed by a Command code which is never equal to SOB value. So duplicating affects the Length, Payload and CRC bytes in frame.
    - Note that CRC computation does NOT include the duplicated SOB character.
- CAN physical transport does not change. CAN uses its own mechanism to transfer the command frame and return the response frame back. CAN does not use the SOB duplication obviously.
- PD-BDM physical transport does not change. The PD-BDM plug-in on PC side uses a BDM direct memory access to upload the command frame and to download a response frame back.

Command/response principle does not change

- Same as before, the protocol is still based on master-slave mechanism. Master sends a command and Slave sends a response. The response must come until a timeout expires.
- The protocol newly supports asynchronous (unsolicited) events to be sent, but this is only possible outside of actual response frame (event messages never "nest" into response frames)
- Response command code still used to represent success (0x00 ... 0x7F) or failure (0x80 ... 0xFF)

New Data Frame Formats

General

- ULEB128 is used to encode address and size values in the protocol, e.g. when accessing memory for reading or writing. This helps the protocol to better accommodate to transports which enable higher MTU. UART/Serial remains 255.
- CRC-8 checksum used instead of simple two's complement.
- More complex CRC algorithms may be used on transports other than UART.
- Transport layer duty is to transfer (COMMAND, LENGTH, PAYLOAD) to the receiver and (STATUS, LENGTH, PAYLOAD) back from receiver. Each transport may use different way to achieve it.

New transports

- This protocol is designed also for Ethernet, UDP or TCP transports. All addresses and sizes are coded as ULEB128, so the transport-related MTU does not affect the content encoding.

UART Transport (and USB-CDC)

UART Transport enables maximum 254 MTU of Payload. The target driver implementation may lower the MTU value to save RAM on small MCUs.

UART Frames

UART Command Frame

- No longer using "fast commands" known in prior versions because length of the payload is not deterministic (that is because addresses and size values used inside the Payload use ULEB-coded numbers)
- Length is always present in all commands and it one byte long (max 254 bytes of data part)

SOB	Command (1 byte)	Length (1 byte)	Payload	CRC (1 byte)
0x2B	Any except 0x2B	**length** of Payload	variable length 0 .. 254	CRC-8
	<---- This part of the frame is covered by CRC ---->			

UART Response frame - Short

- The short response frame has cleared VLEN bit in Status byte (bit6) and no Length field.
- This kind of response is used for any non-error replies (ERR bit clear) from slave to master when master node knows in advance the expected length of the answer (e.g. when reading memory, the answer length is exactly as specified in the Read Memory command).
- Error responses with ERR bit set in the Status byte (bit7) use this Short format **without any Payload** unless the Status bit6 is set.

SOB	Status (1 byte)	Payload	CRC (1 byte)
0x2B	0x00..0xAF & (~0x40)	known length 0 .. 254	CRC-8
	<---- This part of the frame is covered by CRC ---->		

UART Response frame - Long

- The long response with VLEN bit set in Status byte (bit6) contains the length field regardless of ERR bit value.
- This kind of response is used when master does not know the length in advance, for example the Get Configuration Value or when an Error response contains additional information bytes.

SOB	Status (1 byte)	Length (1 byte)	Payload	CRC (1 byte)
0x2B	0x00..0xAF | 0x40	**length** of Payload	variable length 0 .. 254	CRC-8
	<---- This part of the frame is covered by CRC ---->			

UART CRC

UART transport uses **CRC-8-CCITT** implementation as defined on Wikipedia.

Data frame bytes which are equal to SOB value and which are duplicated by UART physical transport layer are only calculated into the CRC value once. The duplicated SOB is NOT calculated to CRC.

UART Throughput Estimation

The following estimations assume the UART communication operates at 115200 bps with one start and one stop bit. Ten bits total are counted for each transferred byte. All calculations below take the following general protocol overhead into consideration:

- Command overhead: **4 bytes** (SOB, Command, Length and CRC).
- Short response overhead: **3 bytes** (SOB, Status, CRC)
- Long response overhead: **4 bytes** (SOB, Status, Length, CRC)
- SOB replication overhead: **1 byte** when SOB byte occurs in payload (probability 1/256). The SOB, Command and Status values are never equal to SOB value.

Reading variable

The Read Memory command uses ULEB-encoded address so the body length varies depending on address value. The size is also ULEB-encoded, however variable size is generally smaller than 127 so the size fits to a single byte.

Example calculation when reading 8 bit value at 32bit address.

- **Command**
  - 4 bytes protocol overhead
  - 5 bytes ULEB-encoded 32bit address
  - 1 byte size
- **Response**
  - 3 bytes protocol overhead
  - 1 byte variable value
- **TOTAL**
  - **14 bytes (+8/256 statistically) ~ 1.22ms ~ 821 transfers/sec**

Typical operations are summarized in the following table:

Read operations	**Byte counts (+SOB replication)**				**UART transactions @115200bps**	
	Cmd.	**Resp.**	**Total**	**SOB**	**Duration**	**Max. rate**
Read 8bit at 32bit address	10	4	14	10/256	1.22ms	820 reads/sec
Read 16bit at 32bit address	10	5	15	11/256	1.31ms	765 reads/sec
Read 32bit at 32bit address	10	7	17	13/256	1.48ms	675 reads/sec
Read 8 bit at 14bit relative address	7	4	11	7/256	0.96ms	1044 reads/sec
Write operations						
Write 8bit at 32bit address	12	3	15	11/256	1.31ms	765 writes/sec
Write 32bit at 32bit address	15	3	18	14/256	1.57ms	638 writes/sec
Oscilloscope - read variables to put to real time graph						
set of 3x 8bit variables	5	6	11	7/256	0.96ms	1044 samples/sec
set of 3x 16bit variables	5	9	14	10/256	1.22ms	820 samples/sec
set of 3x 32bit variables	5	15	20	16/256	1.74ms	574 samples/sec

CAN Transport

CAN transport uses a sequence of up-to 8-byte CAN frames to transfer a full command to target. The first byte of each CAN frame is a signaling byte, the following 7 bytes are used to carry data.

For simplicity of implementation, the CAN transfers commands in the same format as UART, only the SOB byte is omitted. The Command/Status byte, Length byte, Payload and the CRC-8 values are transferred and may be processed the same way as the UART frame.

CAN Frame structure

The FreeMASTER message is split to one or more CAN frames and are transmitted to the CAN bus. Individual CAN frames are not acknowledged by receiver.

All frames have the following structure.

Data byte	Description
0	Control byte
1..7	Data bytes (variable length)

CAN Control byte

CAN control byte describes the CAN frame and enables the receiver to check if all frames were received.

Bit 7	6	5	4	3	2-0
TGL	M2S	FST	LST	SPC	LEN

Bit	Name	Description
7	TGL	Toggle bit. This bit is reset in the first frame and toggled in the following frames. This bit should be checked by receiver to detect a broken frame sequence.
6	M2S	Master-to-slave. This bit is set in FreeMASTER command messages (PC to target) and reset in response messages (target to PC). This bit enables the FreeMASTER CAN protocol to run on the single CAN message ID
5	FST	First frame. This bit is set in the first CAN frame of the FreeMASTER message.
4	LST	Last frame. This bit is set in the last CAN frame of the FreeMASTER message. FST and LST bits may be both set.
3	SPC	Special command (identified by data[1]). Handled by CAN transport, not passed to FreeMASTER decode logic. Currently used for CAN ping only.
2-0	LEN	Number of data bytes present in this CAN frame.

CAN Throughput Estimations

CAN transport encapsulates the UART frame into a sequence of CAN frames. Each CAN frame carries 1 control byte and 7 data bytes of encapsulated "UART" payload. The SOB replication is not applicable in this case. General low-level overhead of CAN physical protocol when using Extended frame format is 100% (128 time-quanta bits are needed to transfer 64 bits of CAN payload).

In our case each 7 useful bytes of encapsulated UART frame require 128 time-quanta bits on the CAN bus. This makes the CAN transmission approximately 1.83x less effective than UART - assuming the bit rates are equal (and counting 10bits for each UART byte).

With CAN bit rate set to a typical value of 500kbps, the CAN throughput is theoretically 2.37x higher than UART at 115200bps.

Direct BDM Communication

When using direct BDM communication over MCU's background debug module or a JTAG, no communication driver is required to run as part of the MCU application. PC host tool is able to read and write any RAM memory location without any intervention of the MCU application code. The FreeMASTER features are limited to plain variable value access. No protocol features like Recorder, TSA or Pipes are available. When any of the features are needed, use the PD-BDM commuication, which still leverages the BDM direct memory access, but ale uses the communication protocol to implement all supported features.

Packet-Driven BDM Communication

In packet driven BDM mode, the same protocol as with Serial line is used. The difference is in the way how frames are physically exchanged. The sender (PC host) initiates the communication by uploading the command frame directly into a target buffer located in the MCU memory. It also uses a dedicated control/status variable to signal the valid command is ready to be processed. Target MCU driver periodically polls the control/status variable and processes the command as soon as it is ready. The response frame is again made available in the memory buffer along with appropriate status value in control/status variable. PC Host downloads the response frame as soon as it finds the status signalled.

There are special constant values used as marks around the target memory buffer in the MCU memory which can be used to locate the communication buffer and the control/status variable automatically by the PC host tool.

Data throughput cannot be estimated when using PD-BDM communication. Depending on the JTAG or BDM probe used, the typical throughput is going to vary between 9600bsp to roughly 50kbps of standard serial communication.

Data Formats

The following data formats are used in the protocol design specification

Format	Byte Size	Description
uint8	1	byte integer
uint16	2	short integer
uint32	4	long integer

uint64	8	long long integer
LEB128	1..N	LEB128-encoded number, not specifically signed or unsigned. Only used in general text to refer to variable-sized number. This document always specifies ULEB128 or SLEB128 when needed.
ULEB128	1..N	unsigned ULEB128-encoded number:
SLEB128	1..N	signed SLEB-128-encoded number
String	N	string formed of a sequence of single-byte ASCII characters
Zero-terminated String	N	string of ASCII characters followed by a zero as a termination
UTF-8	N	string with UTF-8 escape sequences allowed
bytes	N	general array of uint8 bytes

Response Status Codes

Response codes may signal success (ERR=0) or error (ERR=1) result of the processed command. Bits 6 and 5 must be masked off when testing the status Value.

Status code Bit Fields

7	6	5	4	3	2	1	0
ERR	VLEN	EVN	Value				

- **ERR** = (FMSTR_STF_ERR = 0x80): Error Bit: signals an error response. The Short response frame without any Payload is used unless the VLEN bit is also set.
- **VLEN** = (FMSTR_STF_VLEN = 0x40): Variable Length Bit: signals that this response uses the Long response frame with a Length field and a Payload.
- **EVN**= (FMSTR_STF_EVN = 0x20): Event Bit: reserved for future use
- **Value** - Status code value.

Note: Any master node code testing the Response Status Code value should only compare bits masked with 0x9F (only ERR and Value fields). The VLEN and EVN bits should be excluded when evaluating the status code value.

The following table describes possible status values **(masked with value 0x9F)** returned in the Response Frame.

Alias	Code	Description
FMSTR_STS_OK	0x00	General success code
FMSTR_STS_FALSE	0x01	General success code representing a false response.
FMSTR_STC_INVCMD	0x81	Unknown command code (unsupported operation).
FMSTR_STC_CMDCSERR	0x82	Command checksum error
FMSTR_STC_CMDTOOLONG	0x83	Command exceeds MTU, the receive buffer is too small to accept it
FMSTR_STC_RSPBUFFOVF	0x84	The response exceeds MTU, it would not fit into the transmit buffer
FMSTR_STC_INVBUFF	0x85	Invalid buffer length or operation
FMSTR_STC_INVSIZE	0x86	Invalid size specified
FMSTR_STC_BUSY	0x87	Service is busy
FMSTR_STC_NOTINIT	0x88	Service is not initialized
FMSTR_STC_EACCESS	0x89	Access to target resource is denied
FMSTR_STC_EAUTH	0x91	Access to target needs a password authentication
FMSTR_STC_EPASS	0x92	Password authentication failed
FMSTR_STC_EIOCTL	0x93	User resource does not support the Read, Write or IOCTL operation requested.

Commands and Command Codes

Název	Alias	Code	Remarks
Get Configuration Value	FMSTR_CMD_GETCONFIG	0x20	Get configuration parameter value (e.g. MTU, platform name, etc.)
Read Memory	FMSTR_CMD_READMEM	0x21	Read target memory
Read Memory with Base Address	FMSTR_CMD_READMEM_BA	0x22	Read target memory, speed-optimized command
Write Memory	FMSTR_CMD_WRITEMEM	0x23	Write to target memory
Recorder Control	FMSTR_CMD_SETREC	0x24	Configure or control the recorder
Recorder Status	FMSTR_CMD_GETREC	0x25	Get recorder status or configuration
Oscilloscope Control	FMSTR_CMD_SETOSC	0x26	Configure Oscilloscope
Oscilloscope Read	FMSTR_CMD_READOSC	0x27	Read the oscilloscope variables
Pipe Control	FMSTR_CMD_PIPE	0x28	Read/write pipe data
TSA Control	FMSTR_CMD_GETTSAINFO	0x29	Get TSA Table Information
Get string length	FMSTR_CMD_GETSTRLEN	0x2A	Get string length
Authenticate - step 1	FMSTR_CMD_AUTH1	0x2C	Initiate password authentication, request authentication challenge
Authenticate - step 2	FMSTR_CMD_AUTH2	0x2D	Provide authentication data to validate that user knows the password
User Resource Read/Write /IOCTL	FMSTR_CMD_URESRWI	0x2E	User Resource Read, Write or Control
Pipe Information	FMSTR_CMD_GETPIPE	0x2F	Get pipe information
Application Command - Send	FMSTR_CMD_SENDAPPCMD	0x30	Send the Application Command
Application Command - Get Status	FMSTR_CMD_GETAPPCMDSTS	0x31	Get the Application Command status
Application Command - Get Data	FMSTR_CMD_GETAPPCMDDATA	0x32	Get the Application Command data
_Template	FMSTR_CMD_RESERVED	unused: 0x2B	This command is never used

Test Vectors

See protocol test vectors here.

Commands Reference

Název	Alias	Code	Remarks
Get Configuration Value	FMSTR_CMD_GETCONFIG	0x20	Get configuration parameter value (e.g. MTU, platform name, etc.)
Read Memory	FMSTR_CMD_READMEM	0x21	Read target memory
Read Memory with Base Address	FMSTR_CMD_READMEM_BA	0x22	Read target memory, speed-optimized command
Write Memory	FMSTR_CMD_WRITEMEM	0x23	Write to target memory
Recorder Control	FMSTR_CMD_SETREC	0x24	Configure or control the recorder
Recorder Status	FMSTR_CMD_GETREC	0x25	Get recorder status or configuration
Oscilloscope Control	FMSTR_CMD_SETOSC	0x26	Configure Oscilloscope
Oscilloscope Read	FMSTR_CMD_READOSC	0x27	Read the oscilloscope variables
Pipe Control	FMSTR_CMD_PIPE	0x28	Read/write pipe data
TSA Control	FMSTR_CMD_GETTSAINFO	0x29	Get TSA Table Information
Get string length	FMSTR_CMD_GETSTRLEN	0x2A	Get string length
Authenticate - step 1	FMSTR_CMD_AUTH1	0x2C	Initiate password authentication, request authentication challenge
Authenticate - step 2	FMSTR_CMD_AUTH2	0x2D	Provide authentication data to validate that user knows the password
User Resource Read/Write /IOCTL	FMSTR_CMD_URESRWI	0x2E	User Resource Read, Write or Control
Pipe Information	FMSTR_CMD_GETPIPE	0x2F	Get pipe information
Application Command - Send	FMSTR_CMD_SENDAPPCMD	0x30	Send the Application Command
Application Command - Get Status	FMSTR_CMD_GETAPPCMDSTS	0x31	Get the Application Command status
Application Command - Get Data	FMSTR_CMD_GETAPPCMDDATA	0x32	Get the Application Command data
_Template	FMSTR_CMD_RESERVED	unused: 0x2B	This command is never used

Application Command - Get Data

Alias	FMSTR_CMD_GETAPPCMDDATA
Code	0x32
Remarks	Get the Application Command data

Description

This command is used for get the Application Command data.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1..3	ULEB128	data_len	Length of requested data.
2	1..10	ULEB128	data_offset	Offset of requested data.

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VARLEN)

	Byte Size	Format	Name	Description
1	0..N	bytes	data	Requsted data of the Application Command response. Size may be lower than requested data_len when no more data are available at requested data_offset.

Application Command - Get Status

Alias	FMSTR_CMD_GETAPPCMDSTS
Code	0x31
Remarks	Get the Application Command status

Description

This command is used for get the Application Command status.

Data Part

This command carries no data.

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	Format	Name	Description
1	1	uint8	cmd_status	Status of application command.

Application Command - Send

Alias	FMSTR_CMD_SENDAPPCMD
Code	0x30
Remarks	Send the Application Command

Description

This command is used to send an Application Command.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	code	Application command
2	0..N	bytes	args	Command arguments.

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

Authenticate - step 1

Alias	FMSTR_CMD_AUTH1
Code	0x2C
Remarks	Initiate password authentication, request authentication challenge

Description

This command is used to request a random salt as the 1st step when authenticating access with a password. The target provides the identifier of authentication algorithm which needs to be used to generate the access key (see Authenticate - step 2) and provides salt and other challenge data for as the algorithm input.

Authentication Algorithms

Target application implements one of the following authentication algorithms. Future versions of protocol specification and new target MCU drivers may introduce new algorithms and assign them a new ID value. Client must always support all defined authentication algorithms.

No Authentication needed (ID=0)

The 0 is is returned when no password authentication is required for specified access level.

Basic SHA-1 Authentication (ID=1)

1. Target generates salt 16 bytes and sends it to client
2. Client calculates the key as

```
SHA1(salt + SHA1(password) + salt)    // where + means concatenation (sequential hashing of each part)
```

3. Client sends the key back to target.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	access_required	Required access: • 0x00 = reset access to 0. Client sends this command when terminating session. • 0x01 = (R) Read access • 0x02 = (RW) Read+Write access • 0x03 = (RWF) Read+Write+Flash access

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VLEN)

	Byte Size	Format	Name	Description
1	1..5	ULEB128	algo_id	ULEB128-encoded Algorithm ID. Identification of FreeMASTER authentication protocol used. When 0 is returned, no password authentication is needed.
2	N	bytes	salt	Random salt to be used in authentication process
3	*optional*	bytes	...	More challenge parameters provided along with the salt as an authentication challenge

Authenticate - step 2

Alias	FMSTR_CMD_AUTH2
Code	0x2D
Remarks	Provide authentication data to validate that user knows the password

Description

This command is used to prove that the client knows a correct password. This command carries the result hash value computed from the salt obtained by Authenticate - step 1 and from the access password.

Note that the server may support three different passwords, one for each access level R, RW, RWF. The following scenarios may happen:

1. **There is no password needed for requested level and any lower level.** The requested level is granted immediately in Authenticate - step 1. The "step 1" server returns *algo_id=0* so this "step 2" is not taken at all.
2. **Client provides correct password for the requested level:** The requested level is granted successfully.
3. **Client provides password for higher access level:** The *requested* level is granted.
4. **Client provides password for lower access level:** The *lower* level is granted.
5. **Client does not provide valid password:** The AUTH2 command returns an error.

Data Part

This command carries the following data

	Byte Size	**Format**	**Name**	**Description**
1	1	uint8	access_required	Required access (this is the same value as passed into Authenticate - step 1). • 0x01 = (R) Read access • 0x02 = (RW) Read+Write access • 0x03 = (RWF) Read+Write+Flash access
2	N	bytes	access_key	Access key generated by Key Derivation Function as requested by Authenticate - step 1 response earlier

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	**Format**	**Name**	**Description**
1	1	uint8	access_granted	Currently granted access. This is the same value as access_required when password was valid. • 0x01 = R • 0x02 = RW • 0x03 = RWF

Error codes:

- In case the password was invalid or was insufficient for requested access level, the STC_EPASS error code is returned.

Get Configuration Value

Alias	FMSTR_CMD_GETCONFIG
Code	0x20
Remarks	Get configuration parameter value (e.g. MTU, platform name, etc.)

Description

Client uses this command to determine value of a configuration parameter. The parameters are expected to be always constant, defined by the target application. Parameters are named and are accessed by the name or by index value. The name must be unique, most names are defined by this protocol and are required to be supported.

- Accessing by Name is typical when client needs to know a certain value.
- Accessing by index is common to access values indirectly or in a loop.

Rules for naming and indexing:

- Index value 0 is reserved..
- All named parameters must be accessible also by index values starting at 1 going up to N without any gaps. This enables to enumerate all named values by a simple loop.
- Unnamed parameters must be accessible at any index value N+2 or higher so they are not enumerated along with named parameters.
- Index values of unnamed parameters do not need to be consecutive (there may be gaps).

Parameters Defined by Name

Parameter Name	Format	Description
MTU	ULEB128	Size of an internal communication buffer for handling command and response frames. MTU must be at least 32 to enable basic communication. The client never sends commands larger than MTU and never requests data for which the response frame would exceed MTU.
VS	Zero-terminated String	Version string
NM	Zero-terminated String	Application name string
DS	Zero-terminated String	Description string
BD	Zero-terminated String	Build date/time string
F1	uint8	Flags1: • 0x01 - Big Endian Platform. Set to 0=Little Endian and 1=Big Endian. • 0x02 - Enable remote access. When 0, the client must prevent remote hosts to access the target from any remote machine. • 0x04 - reserved • 0x30 - Protection level mask value (2 bits value) • 0<<4 (0x00) - No password required • 1<<4 (0x10) - Password is required to unlock Read access (R) and higher levels • 2<<4 (0x20) - Password is required to unlock Write access (W) and higher levels • 3<<4 (0x30) - Password is required to unlock Write-Flash access (F) level • 0x40 - reserved • 0x80 - reserved
BA	ULEB128	Base Address for optimized FMSTR_CMD_READMEM_BA command and for FMSTR_CMD_WRITEMEM command with BA flag set.
RC	uint8	Number of recorders implemented in system
SC	uint8	Number of oscilloscopes implemented in system
PC	uint8	Number of pipes implemented in system

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1..2	ULEB128	param_index	Index of the parameter required. When 0, the param_name part must follow. When >0 the param_name may be omitted.
2	N	Zero-terminated String	param_name	Parameter name. This member is ignored when param_index is > 0.

Expected Response

When successful the response frame contains the following data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VLEN)

	Byte Size	Format	Name	Description
1	N	Zero-terminated String	param_name	Parameter name
2	N	bytes	value	Data value, format depends on the parameter and must be known to the client so it can properly parse the value.

Get string length

Alias	FMSTR_CMD_GETSTRLEN
Code	0x2A
Remarks	Get string length

Description

This command is used to get string length. It is used for TSA.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1..10	ULEB128	string_addr	Address of string.

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VARLEN)

	Byte Size	Format	Name	Description
1	1..3	ULEB128	string_length	Length of the string.

Oscilloscope Control

Alias	FMSTR_CMD_SETOSC
Code	0x26
Remarks	Configure Oscilloscope

Description

This command is used configure the scope instancies.

Data Part

This command is very universal. A general format of the payload follows:

	Byte Size	Format	Name	Description
1	1	uint8	osc_ix	Oscilloscope instance index value in range 0 to SC-1
2	1	uint8	op_code	Operation code
3	1	uint8	op_length	Operation data length
4	op_length	bytes	op_data	Operation data
5	More "Operations" (fields 2..4) may follow in the same sequence (op_code, op_length, op_data). This enables a single command to perform multiple oscilloscope configuration operations.			

Scope Configuration Operations

Operation Code (op_code)	Name	Description
0x01	Configure oscilloscope memory	Set number of scope variables
0x02	Configure variable	Setup address and size of one oscilloscope variable

Configure Scope Memory (op_code=0x01)

	Byte Size	Format	Name	Description
1	1	uint8	var_count	Number of variables in oscilloscope

Configure Variable (op_code=0x02)

	Byte Size	Format	Name	Description
1	1	uint8	var_ix	Variable index
2	1-10	ULEB128	var_addr	Variable address
3	1	uint8	var_size	Variable size, must be one of standard variable sizes: 1, 2, 4, 8.

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	Format	Name	Description
1	no data			

Oscilloscope Read

Alias	FMSTR_CMD_READOSC
Code	0x27
Remarks	Read the oscilloscope variables

Description

This command is used to read all configured variables of an Oscolloscope instance.

Data Part

This command carries the following data:

	Byte Size	Format	Name	Description
1	1	uint8	osc_ix	Oscilloscope instance index value in range 0 to SC-1

Expected Response

When successful the response frame contains the following fixed-length data (client knows the expected size already):

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	Format	Name	Description
1	size of variable 1	bytes	variable_1	Value of first configured variable
2	...	bytes	...	...
3	size of variable n	bytes	variable_n	Value of last configured variable

Pipe Control

Alias	FMSTR_CMD_PIPE
Code	0x28
Remarks	Read/write pipe data

Description

This command is used to read and write data from pipe and to acknowledge data bytes received previously. Pipe implementation uses transmit buffering to store data until the data are successfully received and acknowledged by the peer.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	port_and_toggle	Pipe port number in lower 7 bits. The MSB bit toggles each time next message is sent. The MSB is used to determine any lost message (when bit value does not match the expected state).
2	1	uint8	bytes_received	Count of bytes previously received by client. This amount may be safely removed from server's transmit buffer.
3	0..N	bytes	pipe_data	Pipe data to be written to the pipe by the client.

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VARLEN)

	Byte Size	Format	Name	Description
1	1	uint8	port_and_toggle	Pipe port number in lower 7 bits. The MSB bit toggles each time next message is sent. The MSB is used to determine any lost message (when bit value does not match the expected state).
2	1	uint8	bytes_received	Count of bytes previously received by server. This amount may be safely removed from client's transmit buffer.
3	0..N	bytes	pipe_data	Pipe data to be read from the pipe by the client.

Pipe Information

Alias	FMSTR_CMD_GETPIPE
Code	0x2F
Remarks	Get pipe information

Description

This command is used to obtain read-only information about a pipe object.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	flags	0x01 - use port instead of index
2	1	uint8	pipe_identifier	Pipe index or pipe port depending on flag 0x01, For indexes value from 0..PC (see Get Configuration Value)
3	1	uint8	cfg_code	One of the configuration codes as described in the table below.

Pipe Configuration Operations

Configuration Code (cfg_code)	Name	Description
0x81	Get pipe name	String name of pipe object
0x82	Get pipe info	Type and formatting information of pipe data

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VARLEN)

Response payload

The response data payload depends on the configuration code (cfg_code) requested:

Get Pipe Description (cfg_code=0x81)

	Byte Size	Format	Name	Description
1	N	String	description	Pipe name/description string

Get Pipe Info (cfg_code=0x82)

	Byte Size	Format	Name	Description
1	1	uint8	port	Pipe port number (value 0..0x7F)
2	1	uint8	type	General information about pipe type and intended usage. The type value bits can be extracted as follows:

<table>
<tr><td></td><td></td><td></td><td></td><td>
<table>
<tr><th>bits 7-4</th><th>3-2</th><th>1-0</th></tr>
<tr><td>reserved bits</td><td>pipe usage/mode

0 = console/terminal
1 = uint dump
2 = sint dump
3 = real dump</td><td>element size (log$_2$)

0 = 1byte
1 = 2bytes
2 = 4bytes
3 = 8bytes</td></tr>
</table>
<p>Terminal types are defined as follows:</p>
<ul>
<li>0x00 represents the ANSI character I/O terminal.</li>
<li>0x01 represents UNICODE wide-character I/O terminal</li>
<li>0x02..0x03 ... reserved values</li>
</ul>
<p>Note that "real" values with element size 1 and 2 bytes are reserved</p>
</td></tr>
<tr><td>3</td><td>1</td><td>uint8</td><td>flags</td><td>0x01 - is open</td></tr>
</table>

Read Memory

Alias	FMSTR_CMD_READMEM
Code	0x21
Remarks	Read target memory

Description

Used when reading variables or other kind of target memory.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1-10	ULEB128	addr	Address of memory to be read.
2	1-2	ULEB128	size	Size of the memory to be read

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	Format	Name	Description
1	size	bytes	data	The memory content as requested

Read Memory with Base Address

Alias	FMSTR_CMD_READMEM_BA
Code	0x22
Remarks	Read target memory, speed-optimized command

Description

Used when reading variables or other kind of target memory. Size of the read command may be significantly optimized by selecting proper Base Address (BA) parameter in the configuration.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1-10	SLEB128	addr_offset	Address of memory to be read specified as a signed offset from Base Address defined by BA configuration parameter.
2	1-2	ULEB128	size	Size of the memory to be read

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	Format	Name	Description
1	size	bytes	data	The memory content as requested

Recorder Control

Alias	FMSTR_CMD_SETREC
Code	0x24
Remarks	Configure or control the recorder

Description

This command is used to configure or control one of the Recorder instances which is implemented in the system.

Data Part

This command is very universal. A general format of the payload follows:

	Byte Size	Format	Name	Description
1	1	uint8	rec_ix	Recorder instance index value in range 0 to RC-1
2	1	uint8	op_code	Operation code
3	1	uint8	op_length	Operation data length
4	op_length	bytes	op_data	Operation data
5	More "Operations" (fields 2..4) may follow in the same sequence (op_code, op_length, op_data). This enables a single command to perform multiple recorder configuration operations.			

Recorder Configuration Operations

Operation Code (op_code)	Name	Description
0x01	Configure recorder memory	Set number of recorder variables, recorder points and pre-trigger points
0x02	Configure variable	Setup address, size, and threshold detection of one recorder variable
0x03	Start Recorder	Start recorder if not yet running
0x04	Stop Recorder	Stop recorder immediately (recorder status may be "no-data" or there may be less data than required when stopped during the initial cycle)

Configure Recorder Memory (op_code=0x01)

	Byte Size	Format	Name	Description
1	1	uint8	var_count	Number of variables in Recorder
2	1-10	ULEB128	rec_points	Number of recorder points used, 0 means maximal possible count of points that fits into the buffer
3	1-10	ULEB128	pretrg_points	Number of "pre-trigger" points to keep in buffer which were recorder before the trigger has occured
4	1-3	ULEB128	time_div	Divisor value of recorder "clock"

Configure Variable (op_code=0x02)

	Byte Size	Format	Name	Description
1	1	uint8	var_ix	Variable index
2	1-10	ULEB128	var_addr	Variable address

3	1	uint8	var_size	Variable size, must be one of standard variable sizes: 1, 2, 4, 8.
4	1	uint8	trg_type	Trigger type and flags: • 0x03 ... mask for variable triggering mode (selection of threshold compare operation) ◦ 0x00 .. this variable not used for triggering ◦ 0x01 .. use as unsigned integer of var_size ◦ 0x02 .. use as signed integer of var_size ◦ 0x03 .. use as floating point value (var_size must be 4 or 8) • 0x04 .. trigger-only, when this bit is set the variable is NOT recorded and is only used for triggering • 0x10 .. trigger when above the threshold • 0x20 .. trigger when below the threshold • 0x40 .. bit clear: normal edge trigger; bit set: level trigger any time the value is above/below threshold • 0x80 .. variable threshold (when set, the trg_thr is an address of variable with the same size)
5	1-10	ULEB128	trg_thr	Trigger threshold value, maximum 8-byte value encoded as ULEB128 When (trg_type & 0x80) is non-zero, then this value is an ULEB128-encoded address of variable used as a trigger threshold. Such threshold variable is expected to be of the same size and same type as this recorder variable.

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

	Byte Size	Format	Name	Description
1	no data			

Recorder Status

Alias	FMSTR_CMD_GETREC
Code	0x25
Remarks	Get recorder status or configuration

Description

This command is used to read status or configuration value of a Recorder instance.

Data Part

This command carries the following data:

	Byte Size	Format	Name	Description
1	1	uint8	rec_ix	Recorder instance index value in range 0 to RC-1
2	1	uint8	cfg_code	Configuration value code

Recorder Configuration Operations

Configuration Code (cfg_code)	Name	Description
0x81	Get recorder description	String description of recorder sampling point etc. (e.g. "PWM Reload Interrupt", or "Timer interrupt")
0x82	Get recorder limits	Get maximum number of recorder variables, and maximum size of the recorder memory in bytes.
0x83	Get recorder info	Get recorder base address, number of recorded variables, and other information needed to download and show recorder graph
0x84	Get recorder status	Get current recorder status (running/stopped etc.)

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VLEN)

Response payload

The response data payload depends on the configuration code (cfg_code) requested:

Get Recorder Description (cfg_code=0x81)

	Byte Size	Format	Name	Description
1	N	String	description	Recorder description string

Get Recorder Memory Limits (cfg_code=0x82)

	Byte Size	Format	Name	Description

1	N	ULEB128	rec_buff_size	Size of raw recorder buffer, the buffer is used for both variable configuration storage and for data recording
2	N	ULEB128	rec_base_rate_ns	Base speed of recorder sampling in nanoseconds. Client may request to sample at integer multiples of this value.
3	N	ULEB128	rec_struct_size	Size of Recorder internal structure. This could be used to compute total samples count on PC side.
4	N	ULEB128	rec_var_struct_size	Size of Recorder variable internal structure. This could be used to compute total samples count on PC side.

Get Recorder Info (cfg_code=0x83)

	Byte Size	Format	Name	Description
1	1	uint8	rec_status	Current recorder status, same as reported by Get Recorder Status (cfg_code=83) - see below
2	1	uint8	var_count	Number of variables configured for recording (the trigger-only variables are excluded!)
3	N	ULEB128	buff_addr	Base address of the recorder buffer
4	N	ULEB128	point_size	Size of one set of sampled values (sum of sizes of all currently recorded variables)
5	N	ULEB128	point_count	Size of currently used recorder buffer in points (used_memory = point_count * point_size)
6	N	ULEB128	point_first	Index of the oldest point in the buffer (i.e. the next write "pointer" when recording in circular buffer)

Get Recorder Status (cfg_code=0x84)

	Byte Size	Format	Name	Description
1	1	uint8	rec_status	Current recorder status: • 0x00 ... not configured • 0x01 ... configured, stopped, no-data • 0x02 ... running • 0x03 ... stopped, not enough data sampled • 0x04 ... stopped, data ready

TSA Control

Alias	FMSTR_CMD_GETTSAINFO
Code	0x29
Remarks	Get TSA Table Information

Description

This command is used to get information about all TSA tables provided by the server application. This command returns an address, size and other information to the client, so the client is able to download and subsequently parse the tables.

TSA Table Structure (TSA version 3)

TSA tables are arrays of fixed-length structure types terminated by an invalid (zero-filled) record. There are 16 bit and 32 bit records, protocol V4 adds new 64 bit record. Record size is identified in tsa_flags.

General entry format, each member is uint16, unit32 or uint64, depending on table record type.

Index	Name	Description
0	name	Object name pointer. An address of Zero-terminated String with name of the object. This may be name of variable, structure data type or special record like memory-mapped directory or file.

1	type	Type name pointer. An address of Zero-terminated String with name of data type.
		
		This name may start with a special (non-printable) character which identifies a native type:
		
		• signed/unsigned integer 1, 2, 4, 8 bytes
		• signed/unsigned fractional number 1, 2, 4, or 8 bytes with custom resolution (UQm.n or Qm.n)
		• floating point number 4 or 8 bytes
		• special character type which identifies files, web-links and other resources (type name follows the initial character)
		
		The special character bits can be described as follows:
		
		`   111STTZZ: where TT=type[int,frac,fp,special] S=signed ZZ=size[1,2,4,8]`
		`   11101100 (0xEC): special ZZ=0: special memory-mapped object (e.g. MEMFILE, PRJ, HREF)`
		`   11101101 (0xED): special ZZ=1: special non-memory mapped object (e.g. DIR, STRUCT, ENUM, CONST, U:xxx)`
		
		In case the name starts with a normal printable character, this is a user-defined type name (e.g. name of the structure type or structure member).
		
		The following 0xEC special type strings are defined (the "addr" field points to the record memory and "info" record contains size and access bits):
		
		• "\xEC:MEMFILE" ... Memory-mapped File entry
		• "\xEC:PRJ" ... Project File Link entry
		• "\xEC:HREF" ... WEB Link entry
		
		The following 0xED special type strings are defined (the "addr" and "info" fields are values with custom meaning):
		
		• "\xED:STRUCT" ... Structure, Union or Class type definition
		• "\xED:ENUM" ... Enumeration type definition
		• "\xED:CONST" ... Named constant as part of enumeration type (entry must follow the ENUM definition)
		• "\xED:DIR" ... Directory entry
		• "\xED:U:FILE" ... User-defined Resource: File
		• "\xED:U:FW" ... User-defined Resource: Firmware Image
		• "\xED:U:PROM" ... User-defined Resource: EEPROM, Flash, or other kind of fixed-size persistent storage

2	addr	This value depends on object type: • **Variable**: variable address Special EC records: • **Memory-mapped File entry**: address of file memory • **Project File Link entry**: address of String with URI of the project location (may be local path or web link) • **Web Link entry**: address of String with web link URI

		Special ED records: • **Structure type**: unused • **Structure member type**: offset of the member within parent type, in bytes • **Enumeration type**: unused • **Enumeration constant**: direct constant value • **Directory entry**: unused • **User-defined Resource**: user-defined handle (e.g. pointer to user's callback function)
3	info	**For variables and EC records:** Contains sizeof(object)<<2. The two LSB bits contain object identification flags: • 0x00...0003 ... entry type mask, see below • 0xFF...FFFC ... size mask (spans from bit 2 up to MSB of the entry) • for 16bit TSA table, the maximum object size is 14 bits (0...16kB) • for 32bit TSA table, the maximum object size is 30 bits (0...1GB) • for 64bit TSA table, the maximum object size is 38 bits (0...256GB), the upper 24 bits are reserved Object identification (lower two bits of the info member): • 0x0 ... structure member • 0x1 ... read-only variable • 0x2 ... read-only variable located in flash, can be written as part of Flash-write access • 0x3 ... read/write variable **For ED records:** • **Structure member type**: sizeof(type)<<2, same as for EC records. Lower two bits are reserved. • **Enumeration type**: sizeof(type)<<2, same as for EC records. Lower two bits are reserved. • **Enumeration constant**: unused • **Directory entry**: unused • **User-defined Resource**: user-defined context data (e.g. context parameter of user's callback function specified in addr)

Difference to older TSAv2 format

This new TSAv3 format introduces new features:

- Introduces 64bit address support, organization of *tsa_flags* value returned by FMSTR_CMD_GETTSAINFO command is different.
- Redefines meaning of Access flags in the *info* member of TSA table.
- Defines new non-memory mapped resources marked with ED special type
- Defines new "Enumeration type" and "Enumeration constant" records to describe C-like "enum" data type which can be referred by variable definitions as their type.
- Defines new "User-defined Resource" object type with application-specific handling.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1..2	ULEB128	table_index	Index of table the PC requests. The server should implement tables starting by index 0 to table_count-1 without gaps.

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VARLEN)

	Byte Size	Format	Name	Description
1	1	uint8	tsa_flags	Contains TSA version, flags and size of TSA table entry items. • 0x0f .. version mask, current version 2 • 0x30 .. table entry size mask • 0x00 .. 16bit table (8 bytes table entries) • 0x10 .. 32bit table (16 bytes table entries) • 0x20 .. 64bit table (32 bytes table entries) • 0x80 .. HawkV2 special addressing mode flag for backward compatibility only
2	1..3	ULEB128	table_size	Size of the table requested by table_index. 0 is returned when the requested table does not exist (end of table list

				reached).
3	1..10	ULEB128	table_addr	Address of the table (0 when table does not exist)
4	*(optional)*0.. N	bytes	tsa_custom	Additional data and flags specific to TSA table version.

User Resource Read/Write/IOCTL

Alias	FMSTR_CMD_URESRWI
Code	0x2E
Remarks	User Resource Read, Write or Control

Description

This command is used to read, write or control any user-defined resource which is not normally mapped to memory. This can be an external EEPROM content, a file located on external file system like SD Card or hard drive, etc.

Client application uses this command to manipulate user resources which exists statically and are assigned a unique identifier (resource_id). The resources may be described either by TSA "User Resource" record, or defined fully on the client's side. In any case, the "resource_id" value is used to uniquely identify the resource and has a form of a pointer value. The value is encoded in ULEB128 format in communication frames. The "resource_id" value is purely numeric value without any special meaning or encoding in the client application. The server implementation in MCU may treat the "resource_id" value as a pointer to local context data or as a plain numerical identifier, depending on MCU application design.

In addition to Read and Write operations, there is a general IOCTL operation to perform non-standard access or control operations identified by an IOCTL code. Set of standard IOCTL codes are defined by this protocol which enables the client-side application to perform "standard" operations like erase, get size, set size, eject, etc. Each IOCTL operation may be assigned context data on input and/or on output. User may define additional IOCTL codes with custom application-specific handling.

Note that the standard protocol driver code on the MCU side does not natively handle any command directly. All Read, Write and IOCTL operations are handled by the application via callback functions from the protocol driver code. This of course enables the application to use any IOCTL operation (including the standard IOCTL operations) to do whatever is needed. It is however strongly discouraged to assign completely different behavior to standard IOCTL codes defined in this specification.

Flash Memory Access

The user-defined resource access may be used to implement Flash Programmer Interface similar to the one supported in older version of FreeMASTER "Classic" tool v2.0 (see specification here: flash_prog.pdf). The Flash interface should support the following standard Read and Write operations and the following IOCTL operations:

- IOCTL_GET_BUSY
- IOCTL_GET_ACCESS
- IOCTL_ERASE
- IOCTL_BLANK_CHECK
- IOCTL_HASH
- IOCTL_GET_BLKINFO

Standard IOCTL Operations

This table list all IOCTL operation codes defined by this protocol specification.

IOCTL code is always encoded as ULEB128 number.

- Range reserved for standard codes is 0 ... 0x7FF. Encoded to single-byte or two bytes ULEB128 value.
- Area for user-defined codes is 0x800...0x1FFF. Encoded to two bytes ULEB128 value
- Area reserved for future use is the 0x2000 and above.

Note that standard code values use even values (bit0 clear) for GET operations and odd values (bit0 set) for SET operations. This enables the access right checking to be also performed when executing this command.

Code	Name	Input Data	Output Data	Description
0x00	IOCTL_GET_BUSY		• uint8 status • uint8 err_code	Determine is resource is ready and able to handle read, write and other IOCTL operations. Possible "status" output values: • 0x00 ... resource is ready, the last operation has finished well • 0x01 ... resource is busy and does not accept any other read/write/IOCTL operations (command would return FMSTR_STC_BUSY) In case the status returns 0x00, the err_code is present and informs about the result of the last operation (one of FMSTR_STC_xxx error codes).
0x01	IOCTL_WRITE_FLUSH			Flush data pending in write cache into the physical device.

0x02	IOCTL_GET_ACCESS		• uint8 access	Get required access level to be able to read, write and IOCTL get/set operations Possible output values: • 0x03 ... mask for **read** operation • 0x0c ... mask for IOCTL **GET** or other non-intrusive operations (code with bit0=0) • 0x30 ... mask for **write** operation • 0xc0 ... mask for IOCTL **SET** or other intrusive operations (code with bit0=1) Each pair of bits encode one of the following values: • 0 ... read level is enough • 1 ... write level must be authorized • 2 ... flash level must be authorized • 3 ... reserved **Default output** values are different for different kinds of user resources (assumed when IOCTL is not implemented): • U:FILE ... 0x50 • U:FW ... 0xaa • U:PROM ... 0x55
0x04	IOCTL_GET_SIZE		• ULEB128	Get current size of resource (e.g. EEPROM size or SD Card File size).
0x05	IOCTL_SET_SIZE	• ULEB128 set_size	• ULEB128 new_size	Set size of resource (e.g. SD Card File size). Returns the new resource size.
0x06	IOCTL_GET_MAX_SIZE		• ULEB128	Get maximum size of a resource for resources which support IOCTL_SET_SIZE
0x07	IOCTL_ERASE	• ULEB128 address • ULEB128 size		Erase portion of the resource content defined by address and size.
0x08	IOCTL_BLANK_CHECK	• ULEB128 address • ULEB128 size		Determine if portion of the resource is erased.
0x0A	IOCTL_HASH	• ULEB128 hash_type • ULEB128 address • ULEB128 length	• bytes hash_result	Calculate hash value of the portion of the resource content. Possible hash_types (server does not need to support all types): • 0x00 ... CRC-8-CCITT • 0x01 ... CRC-16-CCITT • 0x02 ... CRC-32-CCITT • 0x10 ... SHA-1 • 0x11 ... SHA-256 • 0x12 ... SHA-512
0x0C	IOCTL_GET_BLKINFO	• ULEB128 address	• ULEB128 w_base • ULEB128 w_size • ULEB128 e_base • ULEB128 e_size	Determine the base address and size of the writable and erasable block containing given address.

Default Behavior

Implementing IOCTL commands is not mandatory for any resource being accessed. The server application should return FMSTR_STC_EIOCTL when it does not support the requested IOCTL operation code. In this case, the client will assume a default response value of "empty" or zero, unless specified differently in the table above.

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	op_code	Operation to perform and flags - 0x07 ... operation mask - 0x00 ... Read - 0x01 ... Write - 0x02 ... IOCTL - 0x03 ... reserved - 0xF8 ... reserved flags
2	1..10	ULEB128	resource_id	The ID of the resource to perform operation with.
3	N	bytes	data	Data accompanying the requested operation. See next tables for different options

Read Operation Data

The Read Operation command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	op_code	- 0x00 ... Read operation code encoded in lower two bits (mask 0x03) - No flags are defined yet for read operation, bits in mask 0xF8 are all zero
2	1..10	ULEB128	resource_id	The ID of the resource to perform operation with.
3	1..10	ULEB128	read_offset	Zero-based offset to read the resource at
4	1..2	ULEB128	read_len	Length in bytes to read from the resource

Write Operation Data

The Write Operation command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	op_code	- 0x01 ... Write operation code encoded in lower two bits (mask 0x03) - No flags are defined yet for write operation, bits in mask 0xF8 are all zero
2	1..10	ULEB128	resource_id	The ID of the resource to perform operation with.
3	1..10	ULEB128	write_offset	Zero-based offset to write the resource at
4	1..2	ULEB128	write_len	Length in bytes to write to the resource
5	write_len	bytes	write_data	Data to be written

IOCTL Operation Data

The IOCTL Operation command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	op_code	- 0x02 ... IOCTL operation code encoded in lower two bits (mask 0x03) - No flags are defined yet for IOCTL operation, bits in mask 0xF8 are all zero
2	1..10	ULEB128	resource_id	The ID of the resource to perform operation with.
3	1..3	ULEB128	ioctl_code	IOCTL code
4	N	bytes	input_data	IOCTL input data (see standard IOCTL codes in table above)

Expected Response

When successful the response frame contains the following variable-length data:

Status code: 0x40 (FMSTR_STS_OK | FMSTR_STF_VLEN)

	Byte Size	Format	Name	Description
1	N	bytes	output_data	For read operation: • Data provided as a response to read. Output size N is the "read_len" amount of bytes required by the command or it may be less (even 0) if resource reaches an end-of-file condition. For write operation • ULEB128-encoded number of bytes accepted for the write operation. This is the amount required by the "write_len" or it may be less (even 0) if resource reaches an maximum size or other end-of-file condition. For IOCTL operation • Data provided as a response to IOCTL operation. See the "Output Data" column in IOCTL code table above.

Possible error codes

- FMSTR_STC_EIOCTL ... required Read, Write or IOCTL operation is not supported
- FMSTR_STC_BUSY ... resource is currently busy, repeat the operation again

Write Memory

Alias	FMSTR_CMD_WRITEMEM
Code	0x23
Remarks	Write to target memory

Description

Used by the client when writing to target memory including the flash memory (which must be signaled specifically).

Data Part

This command carries the following data

	Byte Size	Format	Name	Description
1	1	uint8	flags	Write flags: • 0x01 ... Write with Mask.The data are followed by an AND-mask value of the same size.
2	1-10	ULEB128	addr	Address of memory to be written.
3	1-2	ULEB128	size	Size of the memory content to be written
4	size	bytes	data	memory content ("size" bytes long) to be written
5	size *(optional)*	bytes	mask	AND-mask for the write ("size" bytes long) - only used when flags indicate the "Write with mask" operation

Expected Response

When successful the response frame contains the following fixed-length data:

Status code: 0x00 (FMSTR_STS_OK)

Byte Size	Format	Name	Description
no data			