FreeEMS Serial Protocol – Core

Version 0.5 - 11th November 2008. (Draft proposal)
GNU General Public License V3 or later.

Foreword

The purpose of the protocol is to provide reliable two-way serial communication with FreeEMS and associated devices over multiple physical protocols. This will commonly be used for live tuning, configuration changes, firmware updates and data acquisition of various types.

This document covers the basic data transfer protocol, which all implementations should adhere to. It describes the methods through which various functions are achieved, right down to the hardware, at all but the application level.

Each implementation will provide documentation on the device-specific functionality offered and the payload data formats used. Not all devices will support all functions, either because they have limited functionality (a display device for example), or because the software versions may differ. Where logging/debug facilities are available, the payload type of unrecognised packets should be noted before discarding them.

Main Attributes

- Compact binary data format to maximize speed and bandwidth.
- Data is packetized with an escaping scheme.
- Each packet, consists of a header, payload and checksum.
- Compulsory integrated low-cost checksum.
- Optional acknowledgment of received packets.
- Big endian multi byte number format (is this correct?)

Contents

Packet Description

Packet Data Format

A packet consists of a three to eight byte header, a payload of variable length from zero to implementation maximum payload size and a one byte checksum. This format will be encapsulated in various lower level constructs depending on the medium of transmission.

Packet Header Layout

Packet Header Specification

Byte Sequence	Length	Data
1	1	Header Flags Bit 0 - Payload type (protocol/firmware) Bit 1 - Acknowledgment valid and/or required Bit 2 - Acknowledgment type (-ve/+ve) Bit 3 – Has Addresses Bit 4 – Has Length Bit 5 - Firmware implementation specific Bit 6 - Firmware implementation specific Bit 7 - Firmware implementation specific
2 and 3	2	Payload ID
4	1 (opt)	Acknowledgment/Sequence Number
5 or 4	1 (opt)	Destination Address
6 or 5	1 (opt)	Source Address
7,8 or 6,7 or 4,5	2 (opt)*	Length Of Payload (excluding header/checksum)

* Required for some packet types

Payload

The payload is specific to the data type being sent. See Payload ID. Data may be Binary or ASCII. ASCII data should be null terminated to allow multiple blocks inside one packet.

Checksum

The checksum is a simple 8 bit type as the EMS deals with bytes during uart transmission. It is compulsory to include the checksum in each packet. It is located at the end of the payload to ease load on the EMS during send and receive operations and keep the code simple and compact. The checksum covers all bytes in the message before any escaping and packetising occurs, including the header.

Header Details

Payload Type

When the payload type bit of the header flags is set (1) the payload type is “protocol” and defined payload IDs should be implemented in all applications. (See Appendix A.)

When the payload type bit of the header flags is clear (0) the payload type is of the type “firmware”. The firmware payload IDs are covered in the FreeEMS Vanilla Specific protocol specification, and are optional.

Addressing Behaviour

- Source Address - The address in the packet header as where the packet has come from.
- Destination Address - The address in the packet header as where the message is to go.
- Broadcast Address - Destination address of zero means that the packet is for all nodes.
- Local Address - The address configured in the device receiving the packet.

The source address value should always be accurate. Packets with zero for the source address should be dropped. Packets with the source address matching our local address should be responded to with a broadcast error packet indicating that a duplicate address is in use. The source address of such an error packet shows which address is duplicated. Such error packets should have a minimum period such that dual addresses do not cause the medium to be flooded with error messages.

Packets with a destination address of zero (broadcast address) should be processed by all nodes. Packets with a destination address that matches our local address should be processed normally. Packets with a destination address that does not match our local address should be ignored without error.

Of course, responses to requests should have the the local address as the source address, and the original packets source address as the destination address if addressing is required.

Acknowledgments

When a device wishes to receive confirmation that what it asked for was actioned appropriately and indeed whether the outcome was successful or not it should set the header bit for acknowledgement valid/required, and populate the acknowledgement field with the next value in the sequence being used. After processing the request the receiving device will respond with the correct payload type/id, the matching acknowledgement number, and the flag set to either success of failure. In the case of failure the body, if present, will be interpretted as an error code. In the case of success the body will either be empty or contain the requested data.

When a packet is sent without the request for an ack reply, and an error occurs, the EMS can still send a response back. If the original packet was addressed, then so will the reply be. If the original packet had no address then the response type will be async error, not the corresponding reply type for the request sent.

Payload ID

The payload ID is a unique identifier number that indicates the functionality to be delivered by a request packet. Depending on the state of the header bit, the payload ID will be interpretted differently. All payload request IDs are to be even and all response packets are to be the ID of the request plus one. See Apendix A for protocol payload ID definitions.

Length Of Payload

Length from the end of the header to the end of the payload NOT including the checksum.

General Information

Flow Control

Upon receiving a packet that requires a response the EMS will be responsible for disabling receipt of further requests by disabling the receive register full interrupt. By this mechanism it can ignore subsequent requests until the current packet is dealt with and the receive buffer free again. The external device should expect messages to be ignored for some short period after a packet is transmitted and delay sending further packets until after that period is over.

RS232 hardware flow control is not used as it is rarely implemented correctly or even at all in common physical implementations

Reliable Transmission

In all cases the external software will be responsible for ensuring unacknowledged messages sent to the EMS are resent. The EMS will work on a “best effort” basis to handle all requests promptly, however that can not be guaranteed. Although the facility for the EMS to request an acknowledgment for a packet is there, it is unlikely to be used for a number of reasons.

Several generic methods of recognising bad data exist :

- Checksum does not match that supplied.
- Packet length in the header (if present) does not match the actual length received.
- Packet is larger than the maximum allowed size.
- The payload length does not match that of the payload ID (where fixed and defined)

In addition, the specifics of how each different physical layer operates will provide further means of recognising bad packet data as will the different payload types.

Packet Size

Packet size is only limited by the size of the payload size field and the specific implementations hardware software setup. Currently we only require ~2k packet size, however up to 64k is supported by the 16 bit size field. This will ensure it stays useful for a longer time and will therefore allow tool re-use too. If larger data blocks need to be sent in some future Free EMS implementation then the data can be packetised down inside this format without much difficulty. Each implementation should provide the facility to request the maximum permissible packet size.

Protocol Layers

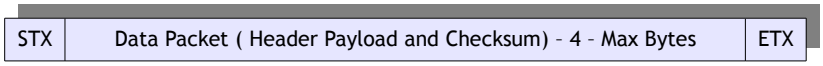
- Physical layers : UART over RS232 or USB, CAN, SPI, I2C etc
- Data layer (UART) : escaping/start/stop/bit format/etc
- Data layer (CAN) : yet to be determined
- Packet layer : generic across all mediums
- Application layer : device/firmware specific

Response Times

To be defined once implemented and tested. Thanks EdG!

UART Specifc Information

Packet Data Format



This is a diagram of a single packet of data, in a serial stream. STX and ETX are single bytes that mark the start and end of the packet (the escaping scheme). Should the data flow be interrupted, the data flow can be picked back up again, by looking for the STX byte. This escaped and delimited packet scheme affords a number of opportunities to catch corrupt data as seen in the Reliable transmission section.

Bitwise Byte Format

Low level 9 bit data format consisting of 11 bits total. One start bit (low), 8 data bits, one hardware generated odd parity bit, one stop bit (high). See MC9S12XDP512V2.pdf section 11.4.3 page 495

Escaping Scheme

Description	Byte Value	Escaped Pair
Start byte (STX)	0xAA	0xBB 0x55
Escape byte (ESC)	0xBB	0xBB 0x44
End byte (ETX)	0xCC	0xBB 0x33

All packets start with STX, and end with ETX. Should any of STX, ETX or ESC occur within the packet contents (including the header and checksum), it must be “escaped”, with a preceding ESC byte and the byte itself XOR'ed with the value 0xFF. This will yield the following easy to recognise pairs in the stream :

If the character 0xBB is found in the raw stream without 0x55, 0x44 or 0x33 following it, the stream and therefore packet is corrupt and invalid and should be discarded.

UART Specific Error Detection

This section is supplimentary to the things mentioned in the reliable delivery section above. The following things should be checked for when interacting over a UART connection of any sort :

- Start byte found before stop byte while inside a packet.
- Escape byte not followed by an escaped special byte.
- Parity bit errors at the byte level.
- Framing errors
- Overrun errors
- Noise errors

Note, not all of these signals will be available on all devices, but whatever is available should be used to ensure a highly robust serial solution is implemented. After all, the EMS code can only confirm that the received data is in good condition, it can not be responsible for the integrity of the data it sends out as it could easily be corrupted in transit.

Note : The checksum doesn't cover the start, stop or escape bytes, only the original packet data.

CAN Specific Information

Yet to be determined.

Appendix A

Payload ID functions and descriptions :

ID	Name	Type	Size	Description And Format
0	Interface Version	Request	0	The unique number that identifies the firmware specific interface definition. If this is the same, there should be absolutely no difference in communications. Useful to allow tuning tools to support interface version ranges based on common functionality and different firmwares with the same interface type.
1	Interface Version	Response	16 - 256	3 unsigned chars, directly followed by an arbitrary string 13 to 253 bytes long. The 3 part version number is only valid for comparison to those with exactly matching string ID sections.
2	Firmware Version	Request	0	The unique string that identifies the firmware version running on the device. Useful for associating behaviours, issues and bugs with specific code versions.
3	Firmware Version	Response	16 - 256	An arbitrary string 16 to 256 bytes long uniquely describing the firmware version running on the device.
4	Max Packet Size	Request	0	Ask the controller to tell us how large in bytes the maximum packet it can handle is. The size is measured from the end of start byte to beginning of stop byte including the header, payload and checksum regions.
5	Max Packet Size	Response	2	16 bit unsigned integer packet size.
6	Echo Packet	Request	Any	Wrap and return the entire packet as the payload.
7	Echoed Packet	Response	Any	The wrapped packet returned.
8	Soft Reset *	Command	0	Instructs the device to software reset itself.
10	Hard Reset *	Command	0	Instructs the device to hardware reset itself. (complete)
13	Error/Exception	Assertion	2	16 bit unsigned integer error number.
15	Debug String	Assertion	Any	Free form null terminated string for debug purposes.

* Note, soft and hard reset behaviour is implementation dependent and may behave the same as each other.

ID : Payload ID number which uniquely identifies the purpose and format of the data contained in the payload.

Type Key :

- Request – A message sent from a tuning device or similar to an embedded device or similar. Typically PC > EMS
- Response – The reply to a request from an embedded device or similar to a tuning device or similar. Typically EMS > PC
- Command - A message sent from a tuning device or similar to an embedded device or similar. Typically PC > EMS
- Assertion - A message sent from an embedded device or similar to a tuning device or similar. Typically EMS > PC

Size : Payload size is in bytes from end of header to beginning of checksum.

Description And Format : The purpose of this particular payload type and how the data is laid out.

Note :

This table is currently incomplete and definitely liable to be extended in subsequent versions.