# FreeEMS Vanilla Serial Interface

*Author:*
FRED COOKE

Version 0.0.3 — Thursday 24th November, 2011

# Contents

## 1 Foreword

This document is based on and assumes knowledge of the FreeEMS Serial Protocol – Core document.

The purpose of this document is to define the application functionality and payload data formats for communicating with FreeEMS Vanilla. 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.

## 2 String Constants

If a payload is described as being a "standard string constant", then it should be parsed in the following way.

- If a "nul" byte is found (`0x00`) then the string terminates, just as in C strings.
- If no "nul" byte is found before the end of the payload then the string consists of the entire payload
- If a "nul" byte is found at an earlier position than the last possible position, then any trailing bytes should be ignored.

## 3 Deprecated

The list below shows all calls which do function, but which are not supported or described here. Do not use them, results could be unpredictable or unreliable. They will be removed from the firmware in future versions.

- `0x0194` Set Async Datalog Type
- `0x012C` Adjust Main Table Cell
- `0x012E` Adjust Main Table RPM Axis
- `0x0130` Adjust Main Table Load Axis
- `0x0132` Adjust 2D Table Axis
- `0x0134` Adjust 2D Table Cell

# 4 Payload Type Definitions

The titles of sub-sections within this section consist of the Payload ID followed by a brief description of the payload type. The body of the section is a detailed description of what it what it does. The structure, length and reply specification follows that and is the last piece of information in each sub-section.

## 4.1 `0x0100` Update Block In RAM

Replaces a block of RAM memory with the provided data. Some blocks are only allowed to be changed as a whole and not partially, IE, size must match the size of the locationID provided and offset must be zero. General configuration blocks can be adjusted using arbitrary offset and size parameters provided that offset is less than size and size is greater than 0 and less than the locationID size. The locationID provided must have non-zero RAM fields.

- Contents: location ID (2), offset (2), size (2), data (1+)
- Length: 7+

## 4.2 `0x0102` Update Block In Flash

Exactly the same as updateBlockInRAM except for flash memory, and updates the corresponding RAM region if there is one. The locationID provided must have non-zero flash fields.

- Contents: locationID (2), offset (2), size (2), data (1+)
- Length: 7+

## 4.3 `0x0104` Retrieve Block From RAM

Exactly as per the title. The locationID provided must have non-zero RAM fields. Size of zero returns entire block.

- Contents: locationID (2), offset (2), size (2)
- Length: 6
- Reply: requestedData (1+)

## 4.4 `0x0106` Retrieve Block From Flash

Exactly as per the title. The locationID provided must have non-zero flash fields. Size of zero returns entire block.

- Contents: locationID (2), offset (2), size (2)
- Length: 6
- Reply: requestedData (1+)

## 4.5 `0x0108` Burn Block From Ram To Flash

Burn data from live tuned RAM down to persistent flash storage. The locationID provided must have non-zero values for all RAM and flash fields.

- Contents: locationID (2), offset (2), size (2)
- Length: 6

## 4.6 `0x0190` Request Basic Datalog

Request a basic datalog packet to be sent back, useful when not streaming. The optional payload contents are the truncation length to be used. If it is not provided the configured truncation length will be used just as in asynchronous mode.

- Contents: Optional Length (2)
- Length: 0 or 2
- Reply: requestedData (1+)

## 4.7 `0x0196` Request Byte LA Datalog

Request a preconfigured single byte of data to be sent back.

- Reply: requestedData (1)

## 4.8 `0x0258` Retrieve Arbitrary Memory

As per the title, for debugging ONLY, could be removed in future.

- Contents: length (2), address (2), RAMPage (1), FlashPage (1)
- Length: 6
- Reply: Original contents (6), requestedData (1+)

### 4.9 `0x6666` Request Unit Test Over Serial

Requests the execution of a device side test which returns a result to be checked on the requesting device, usually a PC with a test suite running. The format is ID and argument data, where the argument data and its format is entirely dependent upon the ID used. One example is provided here, the rest will be found in the test definitions as they are written and added to the project repository. The ID `0x0000` is an empty test which returns its own value as a positive result.

- Contents: Test ID (2), arguments (0+)
- Length: 2+
- Reply: Specified by each Test ID definition.

### 4.10 `0x7777` Start Bench Test Sequence

This function is perfect for both bench and controlled in-car testing of injectors, coils and output circuitry! The current setup is capable of running tests in the range of a single sub-millisecond pulse to a 10 day long-running test! Approximate specifications are listed below.

- Up to 65535 cycles
- Higher than 1kHz max frequency
- Lower than 0.1Hz min frequency
- Single-tick 0.8us granularity

A PC-side user interfaces that implements this protocol could calculate and display the following items and more.

- Cycle Time Period: In micro seconds, milli seconds and/or seconds.
- Test Run Length: in milli seconds, seconds, minutes and/or hours.
- For each PW field: Duty cycle, or NA if a calc mode is chosen (0x01 or 0x02) or OFF if 0 is entered.

The request payload length is at least one, and dependent on the mode found in the first compulsory byte. Currently supported modes are `0x00, 0x01, 0x02` which perform the actions of start, stop and bump, respectively.

#### 4.10.1 Mode `0x00` Stop

This mode stops any test that is currently running with no further outputs occurring after this is received and processed. Output pulses that have already started are allowed to continue until their natural end to preserve the integrity of the test result. The request payload consists solely of the mode byte. If a test is not running it will return an error.

### 4.10.2 Mode `0x01` Start

This mode starts a new test using the included configuration parameters. The payload structure is 24 bytes long (plus headers and footers) and is described below.

- The first byte is the mode, supported modes are listed below, others generate an error.
- The second byte is the number of input events per cycle to perform, the acceptable range is 1 - 255, zero generates an error.
- The third and fourth bytes are an unsigned short which are the number of cycles to execute, 1 - 65535 are acceptable values, zero returns an error.
- The fifth and sixth bytes are another unsigned short which, is the number of 0.8us ticks per event within a cycle, this must be greater than decoder code time.
- The next 6 bytes are the simulated input event to fire on, values greater than the number of input events per cycle are considered off, order is PT2 to PT7.
- The last 12 bytes are interpreted as 6 unsigned shorts, applied to pins in the same order as the last array. The following conditions apply: 0 = channel off, 1 = PW from RefPW, 2 = PW from Dwell, other values less than output ISR code run time return an error, values over that are used as-is. Lastly, if no channels are configured at all, an error will be returned, letting you know that your configuration needs to change.

A full example packet, including headers, checksum and footer, which demonstrates as many aspects as possible, is shown as a hexadecimal byte sequence below.

```
0xAA, 0x00, 0x77, 0x77, 0x01, 0x14, 0x00, 0x10, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x01,
0x00, 0x02, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x13, 0xCC
```

Which breaks down into the following final values and meanings:

- `0xAA` Packet delimiter start byte
- `0x00` Flags
- `0x7777` Payload ID
- `0x01` Mode ”start”
- `0x14` Events per cycle
- `0x0010` Number of cycles
- `0xFFFF` Ticks per event
- `0x00, 0x00, 0x00, 0x00, 0x00, 0x00` - Turn each output on simultaneously on the zeroth event of each cycle
- `0x0100` A very short pulse of 256 0.8us ticks (visible on an SMD LED in daylight)
- `0x0001` Use RefPW for pulse width
- `0x0002` Use Dwell for pulse width
- `0xFFFF` A very long 52ms pulse
- `0x0000, 0x0000` No output, channel disabled
- `0x13` Checksum
- `0xCC` Packet delimiter stop byte

Configuring input event numbers in a valid way is straight forward. For the above packet, valid values to use (for any of the string of 6 zero bytes) are 0x00 to 0x13 inclusive (0 - 19), IE, anything less than the number of simulated input events.

Configuring the pulse with control fields is also straight forward. Please consult the following list for detailed information.

- 0x0000 Channel off
- 0x0001 Use RefPW
- 0x0002 Use Dwell
- 0x0003–Threshold Values lower than the threshold that are not listed above return an error
- Threshold–0xFFFF Values in this range are used as is with a unit value of 0.8 micro seconds
- Values equal to or larger than (ticks per event times events per cycle) will result in a duty cycle of 100%
- Threshold is determined by the implementation and can not be specified here

#### 4.10.3 Mode `0x02` Bump

Bump requests consist of the bump mode byte (0x02) and a value to bump the cycle count by, from whatever it is currently at. If a test is not running it will return an error.

## 4.11 `0xDA5E` Retrieve List Of Location IDs

Retrieves a list of 16 bit/2 byte location IDs falling into different categories depending on the single byte listType argument. 0x00 gets ALL locationIDs. 0x01 gets all locationIDs that match at least one mask bit ”or”. 0x02 gets all locationIDs that match ALL of the mask bits ”and”. See just above the definition of the struct ”blockDetails” in the file structs.h for flag bit details.

- Contents: listType (1), listMask (2)
- Length: 3
- Reply: 16 bit/2 byte locationIDs (0–131072) (always an even number, reality is a much lower count)

## 4.12 `0xEEEE` Request Decoder Name

This call returns the name of the decoder, in standard string constant form, which is precisely the name of the source file that it was built from without the extension. For example ”GM-LT1-CAS-360and8” is obtained by building the decoder source file called ”GM-LT1-CAS-360and8.c”. Knowing this is required to uniquely identify which firmware is installed on a device without ripping the entire image and dissecting it.

### 4.13 `0xEEF0` Request Firmware Build Date

This call returns the time, timezone and date that the firmware was built in the standard unix format such as "Thu Nov 24 14:14:19 CET 2011". This is purely for informational purposes.

### 4.14 `0xEEF2` Request Compiler Version

This call returns the version of GCC which built the firmware. The current tool reports its version in the following way "3.3.6-m68hc1x-20060122". This is purely for informational purposes.

### 4.15 `0xEEF4` Request Operating System

This call returns the name of the operating system on which the firmware was built. An image built on this machine would return the value "Darwin". This is purely for informational purposes.

### 4.16 `0xF8E0` Retrieve Location ID Details

Retrieve the attributes of a location ID. See the definition of the struct "blockDetails" in the file structs.h for flag bit details. Parent is meaningless unless specified in the flags. The pages and addresses are optional.

- Contents: locationID (2)
- Length: 2
- Reply: flags (2), parent (2), RAMPage (1), FlashPage (1), RAMAddress (2), FlashAddress (2), size (2)

### 4.17 `0xFFF0` Clear Counters And Flags To Zero

This call clears various informational and statistical counters and flags that do not affect the operation of the system and provides a clean slate to perform tests of any type without resetting the device and interrupting operation of the engine.

## 5 Copyright