Emus BMS Serial Protocol

# Introduction:

The Emus BMS Control Unit can communicate and interacting with other devices in the system over RS232 or USB interfaces by using a special serial communication protocol. This serial communication protocol is used to send Emus BMS operation status updates to other devices, receive configuration and control messages from other devices, and perform Control Unit firmware updates. This document describes the serial communication protocol to help integrate Emus BMS with other embedded systems.

In order to establish communication with Emus BMS Control Unit over the RS232 or USB interface, the external device must use the following connection settings:

Baud rate: 57.6kpbs;

Data Bits: 8 bits;

Parity: None;

Stop Bits: 1 bit;

Hardware Flow Control: None

# General format

The general format of serial communication is based on text line sentences, separated by carriage return and/or line feed ASCII characters 0x0A or 0x0D, therefore each new text line is considered as a potentially valid sentence. A valid sentence must also satisfy several other rules:

* Sentence name and data fields in text string must be separated by ASCII comma characters as delimiters;
* Sentence name is the first text section until the delimiting comma character;
* The format of sentence name and its data fields must comply with the description in corresponding version of Emus BMS Control Unit firmware documentation;
* Each sentence ends with two characters, that denote the hexadecimal 8-bit value of CRC checksum, which was calculated from all preceding characters in the sentence;
* ASCII characters 0x0A and 0x0D are sentence delimiters, and are not calculated into CRC checksum.

General sentence format notation is:

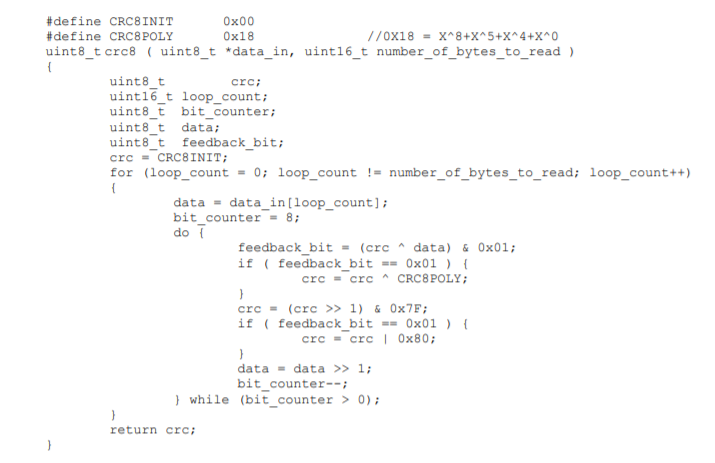
Below is an example of valid sentence text string:



'ST1' is the sentence name. The sentence contains eight data fields, '00' being the first one, and '00040802' being the last. The remaining field '93' is the CRC checksum, calculated from all the preceding characters, starting with 'S' to the final ','. Emus BMS can send the sentences periodically and/or upon request from external device. Usually the request is denoted by '?' in data field of the sentence sent by external device to Emus BMS. Below is an example of request sentence that can be sent to Emus BMS:



'VR1' is sentence name, '?' is the data request symbol, and 'D7' is CRC checksum. Not all sentences support the '?' data request. The details of every specific sentence are described in chapters of this document, dedicated to individual sentences. The CRC checksum is 8 bit value, calculated based on X^8+X^5+X^4+X^0 polynomial with initial value 0. The function example in C programming language that can be used to decode the CRC value is given below:



# Data field encoding types

Data fields in the sentences may have several data coding formats to represent different types of data in simple concise format encoded in text string. Formats are designed for data to be easily encoded or decoded with little processing effort, at the same time requiring modest transmission rate.

## Hexadecimal encoding

Most of the data values are transferred in hexadecimal format, using number characters from '0' to '9' and uppercase characters from 'A' to 'F'. Any data that utilizes this encoding is sent using big endian format, where most significant value comes first. The length of data field that contains hexadecimal value depends on size of value being transferred: 8-bit byte is transferred in 2 characters, 16-bit word is transferred in 4 characters, and 32-bit double word is transferred in 8 characters. There are several types of data that are transferred in hexadecimal encoding, which are described below along with their abbreviations.

**HexCode** - Code of predefined meaning or text, encoded in hexadecimal. Example: if data field has predefined value meanings, such as 0 - Normal, 1 - Warning, 2 - Error, then value '02' in that field means that Emus BMS Control Unit is reporting an error.

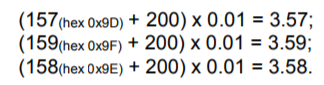
**HexDec** - Fixed point decimal number encoded in hexadecimal. The value is decoded by converting the hexadecimal representation into signed or unsigned integer, adding an offset value, and then multiplying by multiplier (offset and multiplier are specified in this document for each data field where needed). For example, if hexadecimal number '8D' represents unsigned integer with offset of -100 and multiplier of 0.1, then the actual value is:



If the hexadecimal number is '5E', then the actual value would be:



**HexDecByteArray** – An array of bytes that represent fixed point decimal numbers, encoded in hexadecimal. Example: if data field is specified as HexDecByteArray with offset 200 and multiplier 0.01, then a string '9D9F9E' in that field encodes the following values:



**HexBitBool** – A set of logical (Boolean) values, where each bit of a byte represents a separate logical value, encoded in hexadecimal. For example, if data field description specifies that bit 0 represents “Under-voltage” flag, and bit 1 represents “Over-Voltage” flag, then value '02' in that data field means that “Over-voltage” flag is active.

## Decimal encoding

In decimal encoding, a signed or unsigned integers are represented in their usual decimal format.

DecInt – Signed or unsigned integer

## String encoding

Str – Human readable information string

## Sentences:

Consult main pdf for more detail. From the table of contents:

