10920 Madison Ave
Cleveland | Ohio | 44102 | USA
+ 1 216 281 1100
www.Meriam.com
(800) 817-7849

A trusted leader in measurement and calibration solutions.

Meriam Serial Protocol Implementation Guide

For M1500 Digital Transmitters

Referenced from Meriam Serial Protocol for MAP-Based Designs v3.00

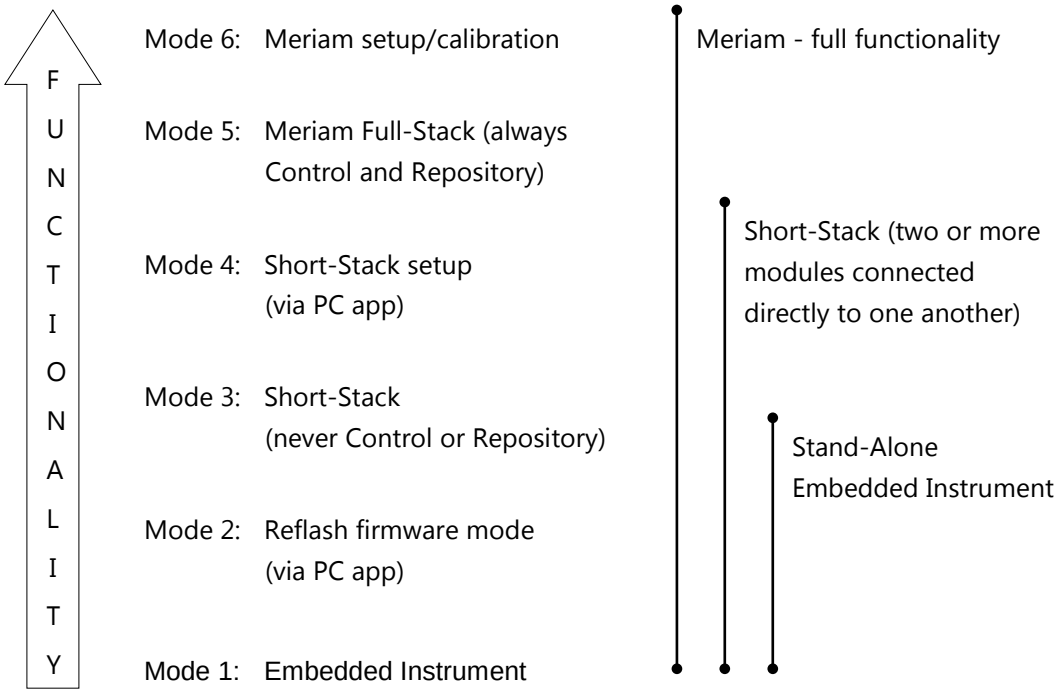
Contents:

Preface

The Modular Architecture Program (now Product) is made up of the following “Classes” and “Types” of “Modules” which can be used alone or in combination to make a finished good.

Class	Type
Measurement / Simulation	EPI (pressure), EVI (volt, mA), EIO (digital IO), EAO (analog out)
Communications / Bridge	RS-232/485, USB
Repository / Data logging	Repository
Control / User Interface	product-specific, M400
Power Supply	product-specific, M400

The diagram below illustrates the communication hierarchy for all classes and types of MAP modules.



The commands are arranged in a hierarchy or “level” of functionality. **Mode 6** contains the most advanced functions and features while **Mode 1** contains the most basic ones. Additionally, each mode inherits all the functions from all the modes below it.

This allows a single module (hardware and firmware) to perform as a stand-alone embedded instrument or be a component of a Meriam instrument.

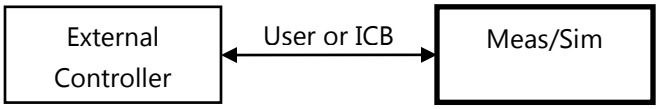
Customers have access to Mode 1.

If a given class/type of module does not support a certain command, an unsupported status will be returned.

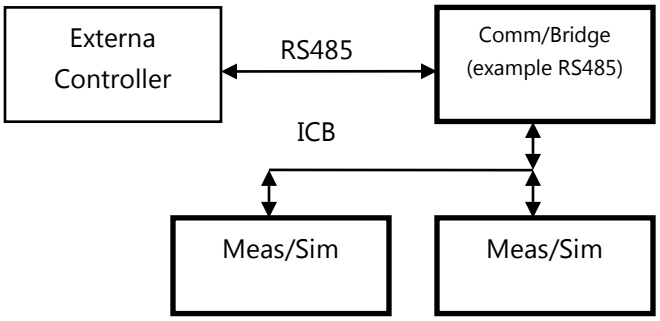
Modular Configurations

The following diagrams show how modules can be combined, from the most basic Embedded Instrument to a Meriam Instrument (also called Full-Stack).

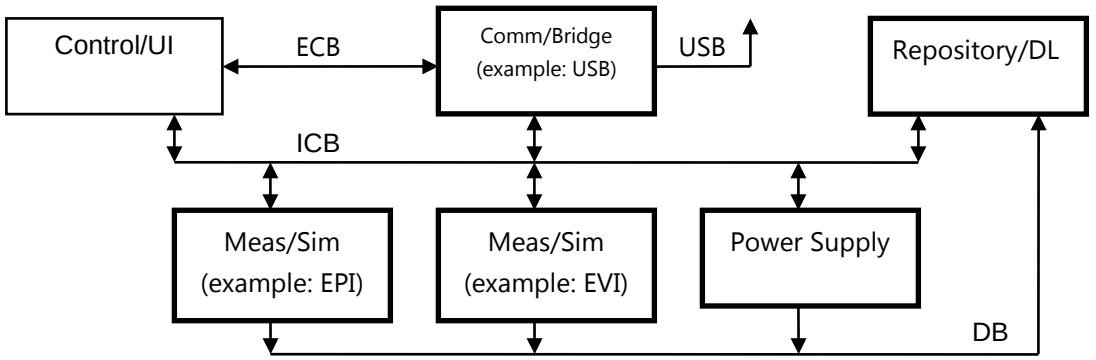
Embedded Instrument



Short-Stack



Full-Stack (a Meriam Instrument, in this case, a M4xx)



Overview

This document describes the message structure and communication protocol between a Controller and an Embedded Instrument (EI) or Short/Full Stack (SS/FS). The Controller is always a Master and the EI is always a Subordinate when used in stand-alone, or Embedded Instrument mode.*

An EI supports three hardware communication interfaces via the 20-pin Meriam Comm. Header (MCH):

- I2C – pins 3-4
- UART – pins 15-16
- SPI – pins 15-18

A SS/FS supports several hardware communication interfaces via the attached comm. board's connector:

- RS232 – RS232485 comm. board connector
- RS485 – RS232485 comm. board connector
- USB2.0 – USB20 comm. board connector

For consistency and ease of interface, the same message structure and protocol is supported across ALL hardware interfaces. This commonality greatly simplifies communication code.

* SS and FS configurations operate both I2C busses in multi-master mode.

Message Structure

A message consists of two basic parts:

- Header including CRC (fixed at 12 bytes)
- Data (variable length)

The fixed-length Header contains basic information about the message, including its length and CRC. The Data portion of the packet (or payload) contains the message-specific, variable-length data, and extended addressing if applicable.

All data (larger than one byte) is little-endian.

This structure facilitates the use of DMA for message reception (that is, the firmware design can take full advantage of the MSP430's USCI/DMA hardware).

This message structure is valid for I2C, UART, and SPI. Although the low-level transmit/receive firmware will be unique for each hardware interface, the message handler will be common.

Communication Protocol

There are two messages types:

- Command
- Response

The Command message is sent from the Controller (the Master) to the EI or SS/FS (the Subordinate). This Command message evokes (or solicits) a Response message.

The commands and responses are intentionally very compact to minimize protocol overhead.

Terminology

Transaction = an exchange of information between a Master and Subordinate

- The Master transmits a Command Message and reads (SPI) or receives (UART and I2C internal/external control busses) a Response Message.

Master = the side that initiates communication

- The Master is not typically able to receive an unsolicited command message (if there is only one Master, this would be a protocol violation).
- The Master is not typically able to receive an unsolicited response message.

Subordinate = the side the responds to communication

- The Subordinate is almost always ready to receive an unsolicited command message.
- The Subordinate is not typically able to receive an unsolicited response message.

Message = a complete "packet" of information (control information and user data (also known as payload) – per Wikipedia)

- A Message is composed of a fixed-length header and a variable length data area
- The Master transmits a Command Message. The Subordinate composes a Response Message

Command Message = a Command Header (CH) followed by Command Data (CD)

- CH = Command Header – 12 bytes of "command message description" info
- CD = Command Data (or Payload) – 0 to 144 bytes of data

Response Message = a Response Header (RH) followed by Response Data (RD)

- RH = Response Header – 12 bytes of "response message description" info
- RD = Response Data (or Payload) – 0 to 144 bytes of data

Normal Message Addressing = addressing for use within a Short/Full-Stack

- Source = 1 byte: typically Module address specified in the CH and RH
- Destination = 1 byte: typically Module address specified in the CH and RH

Extended Message Addressing = addressing required to externally access (For example: PC app, and so on) a Short/Full-Stack, 6 bytes concatenated to the end of the actual data in the Data (or Payload) area

- Source = 3 bytes: Network, Bridge, and Module address
- Destination = 3 bytes: Network, Bridge, and Module address

A Transaction between a Master and Subordinate must be completed (that is, closed) before the Master can initiate another transaction to the same or a different Subordinate. This is true for both Normal addressing (that is, comm. within the stack) and Extended addressing (that is, comm. in/out of the stack) Messages.

Command Format - from "Controller" to Module

Header

1	PRE1	Preamble1	= 0x80	version 1 = 0x80
2	PRE2	Preamble2	= 0x0?	version 1: 0x00 = normal addressing 0x01 = extended addressing ***note:*** *0x01 may map to 0x03 internally, however,* the user will only specify a 0x00 or 0x01
3	LEN	Length	= 0x??	length of DATA area ***note:*** *does NOT include extended addressing*
4	SADD	Srce Address	= 0x??	source (transmitter) address
5	DADD	Dest Address	= 0x??	destination (receiver) address ***note:*** *these addresses describe the current link/hop*
6	CMD1	Command1	= 0x??	main command
7	CMD2	Command2	= 0x??	command argument
8	CMD3	Command3	= 0x??	command argument
9	STAT	Status	= 0x00-0xFF	version 1 = repurposed to command attribute suppress response (SR) bits: 1xxx xxxx = SRc = for this command x1xx xxxx = SRf = set SRc bit on forwarded command ***note:*** *SRf is only processed by comm. boards*
10	CNTR	Counter	= 0x00	version 1 = spare
11	CRCL	CRC16 (LSB)	= 0x00-0xFF	CRC16 of above 10 bytes and Data area
12	CRCH	CRC16 (MSB)	= 0x00-0xFF	(that is, PRE1 thru CNTR and DATA area, inclusive)

***** Unused bytes should be set to 0x00 *****

Data (Payload)

13-n	DATA	Data	=	command-specific, variable-length data
	plus optional extended addressing, which describes the complete address of the command originator and the complete address of the intended command recipient			
n+1	SNET	SrceNetwork	= 0x00-0xFF	source network address
n+2	SBRI	SrceBridge	= 0x10-0x70	source bridge address
n+3	SMOD	SrceModule	= 0x10-0x70	source module address
n+4	DNET	DestNetwork	= 0x00-0xFF	destination network address
n+5	DBRI	DestBridge	= 0x10-0x70	destination bridge address
n+6	DMOD	DestModule	= 0x10-0x70	destination module address

***** Unused bytes (within Length) should be set to 0x00 *****

Data types used to describe the DATA (or Payload) part of the message:

U8	(1 Byte)	= unsigned 8-bit
S8	(1 Byte)	= signed 8-bit
U16	(2 Bytes)	= unsigned 16-bit, little-endian
S16	(2 Bytes)	= signed 16-bit, little-endian
U32	(4 Bytes)	= unsigned 32-bit, little-endian
S32	(4 Bytes)	= signed 32-bit, little-endian
F32	(4 Bytes)	= 32-bit IEEE float, little-endian

Response Format – from Module to "Controller"

Header

1	PRE1	Preamble1	= 0x40	version 1 = 0x40
2	PRE2	Preamble2	= 0x0?	version 1: 0x00 = normal addressing 0x01 = extended addressing
3	LEN	Length	= 0x??	length of DATA area ***note:*** *does NOT include extended addressing*
4	SADD	Srce Address	= 0x??	source (transmitter) address
5	DADD	Dest Address	= 0x??	destination (receiver) address ***note:*** *these addresses describe the current link/hop*
6	CMD1	Command1	= 0x??	echoed
7	CMD2	Command2	= 0x??	echoed
8	CMD3	Command3	= 0x??	echoed (or result)
9	STAT	Status	= 0x00	general status, see status page
10	CNTR	Counter	= 0x00	version 1 = spare
11	CRCL	CRC16 (LSB)	= 0x00-0xFF	CRC16 of above 10 bytes and Data area
12	CRCH	CRC16 (MSB)	= 0x00-0xFF	(that is, PRE1 thru CNTR and DATA area, inclusive)

Data (Payload)

13-n	DATA	Data	=	command-specific, variable-length data
	plus optional extended addressing, which describes the complete address of the response originator (that is, intended command recipient) and the complete address of the response recipient (that is, command originator)			
n+1	SNET	SrceNetwork	= 0x00-0xFF	source network address
n+2	SBRI	SrceBridge	= 0x10-0x70	source bridge address
n+3	SMOD	SrceModule	= 0x10-0x70	source module address
n+4	DNET	DestNetwork	= 0x00-0xFF	destination network address
n+5	DBRI	DestBridge	= 0x10-0x70	destination bridge address
n+6	DMOD	DestModule	= 0x10-0x70	destination module address

What Follows

The next several pages describe the currently supported user commands and their responses.

Only the command-specific header and data bytes are shown. The rest of the header must be populated properly (preamble, addressing, CRC, and so on) as shown on the preceding pages. Unused bytes within the message should be zeroed.

A support file, **EIProtocol.h**, contains defines and data structures/unions that support the following commands (see Appendix A).

Contact Meriam support to obtain a copy of the latest version.

CMD_RESET (0x00)

Command Format - from "Controller" to Module:

LEN	Length	= 0x00	length of DATA area	
CMD1	Command1	= 0x00		
CMD2	Command2	= 0x00-0xFF	Types of reset:	
			0x00 = complete reset, soft reboot	
			0x01 = complete reset, hard reboot	future
			0x02 = comm. buss resets???	future
			0x10 = abort current command	future
			0x11 = abort current action, return to measure	future

Response Format – from Module to "Controller":

LEN	Length	= 0x01	length of DATA area
CMD1	Command1	= 0x00	echoed
CMD2	Command2	= 0x00-0xFF	echoed
STAT	Status	= 0x00-0xFF	general status, see status page
U8	Status*	= 0x00-0xFF	individual status, see status page
	* may not be present for soft reboot (length will be 0 if not present)		

Example	
To …	
	Controller--->EI:
	EI--->Controller:

CMD_GET_SET_INFO (0x02) – constantly adding new CMD3s

Command Format - from "Controller" to Module:

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x02		
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with attributes:	
			xxx1 xxxx = spare	
			xx1x xxxx = spare	
			x1xx xxxx = error (memory = 0)	error is future
			1xxx xxxx = set (get = 0)	
			Lower nibble not bit-encoded:	
			xxxx 0000 = normal (EI) list	
			xxxx 1111 = legacy memory list	future
CMD3	Command3	= 0x00-0xFF	reference number	
U8	Data Bytes	=	data for a set (write)	

Response Format – from Module to "Controller":

LEN	Length	= 0x??	length of DATA area
CMD1	Command1	= 0x02	echoed
CMD2	Command2	= 0x00-0xFF	echoed
STAT	Status	= 0x00-0xFF	general status, see status page
U8	Status*	= 0x00-0xFF	individual status, see status page
U8	Data Bytes	=	data for a get (read)

Example	
To …	
	Controller--->EI:
	EI--->Controller:

Types of information

Normal Reference:

Memory

0x00 = main summary (get): SNs, class, type, addresses, and so on.

0x40 = sensor summary (get): replaces 0x11/0x21, all channels and modes supported

0x80 = main and sensors summary (get): module and main sensor summary

0xC0 = user defined area (get/set): product description

0xC1 = user defined area (get/set): product tag name and asset number

0xC2-0xCF reserved for user (customer) data

ADD HEALTH/DIAGNOSTICS INFORMATION HERE…

Example structure for 0x80 (always refer to EIProtocol.h for up-to-date structures):

```
struct
{                                   //main EE info
    Byte  stat;                     //individual status, see status page
    Byte  pad;                      //spare to align on Word boundary
    char  szStackSN[12];            //stack serial number (for final product)
    char  szModuleSN[12];           //module serial number (for module)
    Word  wProductID;               //product ID number
    char  szProductRev[8];          //product revision
    char  szProductName[32];        //product name

    struct                          //sensor EE(s) info
    {                               //all data here in currently selected eng. units
        char  szSensorSN[12];       //sensor serial number (for sensor)
        float fLSL;                 //lower sensor limit
        float fUSL;                 //upper sensor limit
        char  shorttext[7];         //short units text string (example "inW20C")
                                    //to 6 characters + NULL, left-justified
        Byte  bUnits;               //units index

    } chan[2];

} r80;                              //response r80, 124 bytes
```

CMD_GET_SET_UNITS (0x03)

Command Format - from "Controller" to Module:

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x03		
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with channel:	
			xxx1 xxxx = channel 1	
			EPI (Pressure): measure P1 pressure	
			EVI (Volt Amp): meas/sim volts	
			PS (M400): meas/sim HV volts	
			xx1x xxxx = channel 2	
			EPI (Pressure): measure P2 pressure	
			EVI (Volt Amp): meas/sim milliamps	
			PS (M400): meas Bus Vcc volts	
			x1xx xxxx = channel 3	
			ExI (that is, all): spare	
			PS (M400): meas Battery percent charge	
			1xxx xxxx = channel 4:	
			ExI (that is, all): measure internal temperature	
			Lower nibble not bit-encoded:	
			xxxx 0000 =	get current engineering unit(s) for the specified channel(s)
			xxxx 0001 =	set specified engineering unit(s) for the specified channel(s)
			xxxx 0010 =	read specified engineering unit(s) data for the specified channel(s)
			notes:	
			• the last option (read) can be used to enumerate a complete list of all supported engineering units, one at a time, (via repeated calls) without changing any settings	
			• this command is mode-dependent; it will get/set/read the units for the active mode (that is, meas or sim)	
U8	Unit*	= 0x00-0xFF	value depends upon Command2, for: xxxx 0000, U8 is a don't care xxxx 0001, U8 specifies engineering unit index to set xxxx 0010, U8 specifies engineering unit index to read	

* repeated in groups of 1 byte based upon the number of channels selected in CMD2

CMD_GET_SET_UNITS (0x03) (continued)

Response Format – from Module to "Controller":

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x03	echoed	
CMD2	Command2	= 0x00-0xFF	echoed	
STAT	Status	= 0x00-0xFF	general status, see status page	
U8	Status**	= 0x00-0xFF	individual status, see status page	
U8	Unit**	= 0x00-0xFF	for get	
			• the current unit for the specified channel	
			for set:	
			• if specified unit was valid, the specified unit	
			• if specified unit was invalid, the current unit	
			for read:	
			• if specified unit was valid, the specified unit	
			• if specified unit was invalid, the Nth (last) unit	
S8	Max. LOD**	= 0x00-0xFF	worst-case digits to left of decimal	
S8	Max. AROD**	= 0x00-0xFF	worst-case digits to right of decimal to show accuracy	
S8	Max. RROD**	= 0x00-0xFF	worst-case digits to right of decimal to show precision	
			notes:	
			• the above xODs are worst-case/greatest for the unit	
			• if either ROD is < 0, scientific notation is required to display data correctly	
U8	Spare**	= 0x00	spare to align to Word boundary	
U8	Unit Text**	= 0x00-0xFF	short units text string (For example: "inW20C")	
U8	| **	= 0x00-0xFF	| up to 6 characters + NULL, left-justified	
U8	| **	= 0x00-0xFF	|	
U8	| **	= 0x00-0xFF	|	
U8	| **	= 0x00-0xFF	|	
U8	| **	= 0x00-0xFF	|	
U8	| **	= 0x00	| always a NULL	
U8	Spare**	= 0x00	spare to align to Word boundary	
F32	Conversion**	=	conversion coefficient (PSI to specified engineering unit)	

** repeated in groups of 18 bytes based upon the number of channels selected in CMD2

Example	
To …	
	Controller--->EI:
	EI--->Controller:

EPI Pressure Channel Units:

0	PSI	17	mHg0C
1	inW20C	18	cmHg0C
2	inW4C	19	mmHg0C
3	inW60F	20	torr
4	ftW20C	21	kg/cm2
5	ftW4C	22	kg/m2
6	ftW60F	23	Pa
7	mmW20C	24	hPa
8	mmW4C	25	kPa
9	mmW60F	26	MPa
10	cmW20C	27	Bar
11	cmW4C	28	mBar
12	cmW60F	29	ATM
13	mW20C	30	oz/in2
14	mW4C	31	lb/ft2
15	mW60F	32	User 1*
16	inHg0C	33	User 2*

* measurements are returned in PSI (unless sensor EE factory programmed otherwise), conversions must be done outside of EPI

EVI Volt Channel and Current Channel Units:

EAO Volt Channel and Current Channel Units:

0 mA DC

1 V DC

MAP Internal Temperature Channel Units:

0 °F

1 °C

2 K

3 °R

Note: *future versions may include Universal Units (that is, %) and/or Standard Units (that is, HART or FCINTF, which are a subset of the above units)*

CMD_GET_MEAS (0x04)

Command Format - from "Controller" to Module:

LEN	Length	= 0x00	length of DATA area	
CMD1	Command1	= 0x04		
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with channel:	
			xxx1 xxxx = channel 1	
			EPI (Pressure): measure P1 pressure	
			EVI (Volt Amp): meas/sim volts	
			PS (M400): meas/sim HV volts	
			xx1x xxxx = channel 2	
			EPI (Pressure): measure P2 pressure	
			EVI (Volt Amp): meas/sim milliamps	
			PS (M400): meas Bus Vcc volts	
			x1xx xxxx = channel 3	
			ExI (that is, all): spare	
			PS (M400): meas Battery percent charge	
			1xxx xxxx = channel 4:	
			ExI (that is, all): measure internal temperature	
			Lower nibble not bit-encoded:	
			xxxx 0000 =	gets measurement float* for specified channel(s)
			xxxx 0001 =	same as 0000, but also resets min and max floats (to the current measurement) for specified channel(s)
			xxxx 0010 =	same as 0000, but also gets min and max floats for specified channel(s)
			xxxx 0011 =	spare
			xxxx 0100 =	gets 2 measurement percentage floats*, one relative to LSL/USL and one relative to LRV/URV for specified channel(s)
			note:	
			• this command is mode-dependent; it will get the measurements for the active mode (that is, meas or sim)	

* measurement may be filtered/damped, depending on respective user settings

CMD_GET_MEAS (0x04) (continued)

Response Format – from Module to "Controller":

LEN	Length	= 0x00	length of DATA area	
CMD1	Command1	= 0x04	echoed	
CMD2	Command2	= 0x00-0xFF	echoed	
STAT	Status	= 0x00-0xFF	general status, see status page	
DATA			depends upon lower nibble of CMD2 as shown next	
CMD2 = xxxx0000, xxxx0001				
U8	Status	= 0x00-0xFF	individual status, see status page	
S8	AROD	= 0x00-0xFF	meas-specific digits to right of decimal to show accuracy	
S8	RROD	= 0x00-0xFF	meas-specific digits to right of decimal to show precision	
			note:	
			• the above xODs are signed Bytes • the above xODs are actual/meas-specific for the unit • if either ROD is < 0, scientific notation is required to display data correctly	
U8	Spare	= 0x00	spare to align Float on Word boundary	
F32	Measurement	=	measurement data, 32-bit float in little-endian	
CMD2 = xxxx0010				
U8	Status	= 0x00-0xFF	individual status, see status page	
S8	AROD	= 0x00-0xFF	meas-specific digits to right of decimal to show accuracy	
S8	RROD	= 0x00-0xFF	meas-specific digits to right of decimal to show precision	
U8	Spare	= 0x00	spare to align Float on Word boundary	
F32	Measurement	=	measurement data, 32-bit float in little-endian	
F32	Minimum	=	minimum meas data, 32-bit float in little-endian	
F32	Maximum	=	maximum meas data, 32-bit float in little-endian	
CMD2 = xxxx0011				
spare				

CMD2 = xxxx0100			
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
F32	Percent Limits	=	percent relative to LSL/USL, 32-bit float in little-endian
F32	Percent Range	=	percent relative to LRV/URV, 32-bit float in little-endian

Example	
To …	
	Controller--->EI:
	EI--->Controller:

CMD_MEAS_SIM_MODE (0x05)

Command Format - from "Controller" to Module:

LEN	Length	= 0x08	length of DATA area	
CMD1	Command1	= 0x05	echoed	
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with attributes:	
			xxx1 xxxx = spare xx1x xxxx = spare x1xx xxxx = spare 1xxx xxxx = set (get = 0)	
			Lower nibble not bit-encoded:	
			xxxx 0000 = default	
U8	Mode*	= 0x00-0xFF	the desired measure/simulation mode	
U8	Unit*	= 0x00-0xFF	the desired engineering unit (for mode)	future
F32	Simulation*	=	the desired simulation value (for mode)	
U8	Spare*	= 0x00	spare	
U8	Spare*	= 0x00	spare	

* these parameters are not used (and need not be present) for get

Response Format – from Module to "Controller":

LEN	Length	= 0x0A	length of DATA area	
CMD1	Command1	= 0x05	echoed	
CMD2	Command2	= 0x00-0xFF	echoed	
STAT	Status	= 0x00-0xFF	general status, see status page	
U8	Status	= 0x00-0xFF	individual status, see status page	
U8	Spare	= 0x00	spare to align on Word boundary	
U8	Mode	= 0x00-0xFF	the current measure/simulation mode	
U8	Unit	= 0x00-0xFF	the current engineering unit (for mode)	
F32	Simulation	=	the current measurement/simulation value (for mode)	
U8	State	= 0x00-0xFF	the current hardware state of the power source	
U8	Spare	= 0x00	spare	

Example	
To …	
	Controller--->EI:
	EI--->Controller:

Measure/Simulation Modes by Module:

EVI

0x00 = measure mode: voltage in V*

0x10 = measure mode: current in mA*

0x20 = measure mode: current in mA* (for factory calibration only)

0x30 = simulation mode: source 24V loop* (may be different than precision source V)

0x40 = simulation mode: source voltage in V*

0x50 = simulation mode: source current in mA*

0x60 = simulation mode: sink current in mA*

* the engineering units cannot be changed for any of these modes gaps left for future use (ranges, and so on)

EAO

0x40 = simulation mode: source voltage in V* (4-wire, dedicated power)

0x60 = simulation mode: sink current in mA* (4-wire, dedicated power)

0x70 = simulation mode: sink current in mA* (2-wire, loop power)

* the engineering units cannot be changed for any of these modes gaps left for future use (ranges, and so on)

M4xx Power Supply

0x00 = high voltage off (see State for cause, cannot measure or source in this Mode)

0x40 = high voltage on (Vbat <= HV <= 25.5V, see State for functionality)

The Mode is supplemented by State, to identify and correct various problems.

State (entirely bit-encoded)

Upper nibble is High Voltage State

1xxx xxxx = unused

x1xx xxxx = on (0 = off)

xx1x xxxx = tripped, over-current (0 = not tripped)

xxx1 xxxx = no power available (0 = controlled by command)

Lower nibble is Low Voltage State

xxxx 1xxx = sufficient power to source (0 = insufficient power to source)

xxxx x1xx = sufficient power to measure (0 = insufficient power to measure)

xxxx xx1x = internal power available, isolated (0 = no internal power)

xxxx xxx1 = external power connected, non-isolated (0 = no external power)

NOTICE

This command is not supported on measure-only modules.

CMD_FIELD_RECAL (0x06)

Command Format - from "Controller" to Module:

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x06		
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with channel:	
			xxx1 xxxx = channel 1	
			EPI (Pressure): measure P1 pressure	
			EVI (Volt Amp): measure volts	
			xx1x xxxx = channel 2	
			EPI (Pressure): measure P2 pressure	
			EVI (Volt Amp): measure milliamps	
			x1xx xxxx = channel 3	
			ExI (that is, all): spare	
			1xxx xxxx = channel 4:	
			ExI (that is, all): measure internal temperature	
			notes:	
			• only one channel supported at a time, except for zero	
			Lower nibble not bit-encoded:	
			xxxx 0000 =	zero, offset, tare the offset/tare=future specified channel(s)
			xxxx 0001 =	restore factory defaults
			xxxx 0010 =	get supported field recal procedures future
			xxxx 0011 =	start field recal
			xxxx 0100 =	save point
			xxxx 0101 =	next point
			xxxx 0110 =	finish field recal
			xxxx 0111 =	state field recal

DATA			depends upon lower nibble of CMD2 as shown next

CMD2 = xxxx0000			
CMD3	Command3	= 0x00-0xFF	the operation to perform on the specified channel(s):
			0x00 = zero
			0x01 = offset to specified value (Meas = Value)
			0x02 = offset by specified value (Meas = Meas + Value)
			0x03 = tare to specified value (Meas = Value)
			0x04 = tare by specified value (Meas = Meas + Value)
			0x10 = zero limits future
			0x11 = offset limits future
			0x12 = tare limits future
			notes:
			• zero and offset are non-volatile
			• tare is volatile (that is, reset to off on power on)
			0x60 = EVI zero macro (zeros all 6 modes)
			notes:
			• the electrical inputs must be shorted for the duration of this operation
			• this is a special “macro” command for the EVI only
			• this operation takes several seconds to complete
F32	Value*	=	value in currently selected user engineering units

* repeated in groups of 1 float based upon the number of channels selected in CMD2 this parameter is not used (and need not be present) for zero (or EVI zero)

CMD2 = xxxx0001			
CMD3	Command3	= 0x00-0xFF	Upper nibble spare:
			Lower nibble not bit-encoded:
			xxxx 0000 = restore all points and field-recal
			xxxx 0001 = restore all points
			xxxx 0010 = restore field-recal

NO DATA (just command header)	

CMD2 = xxxx0010			
Future…			

CMD2 = xxxx0011(start field recal)			
U8	Recal Number	= 0x00-0xFF	field recal number (from “get supported FR”)
U8	View/Perform	= 0x00-0xFF	0x00 = View recal, 0x01 = Perform recal

CMD2 = xxxx0100 (save point)			
U8	Recal Number	= 0x00-0xFF	must be same value used in “start FR”
U8	View/Perform	= 0x00-0xFF	must be same value used in “start FR”
U8	Cur Point	= 0x00-0xFF	must be same value returned from “start FR”/“next point”
U8	Num Points	= 0x00-0xFF	must be same value returned from “start FR”/“next point”
F32	Apply Point*	=	for View: the new Apply Point for Recal: the actual “applied” point

* exactly the Apply Point or between the Min/Max Apply Points, returned from “start FR”/“next point”

- **for View and Recal:** if attempting to save a point outside Min/Max Apply Points, the EI response will return an error
- **for Recal**: if attempting to save a point outside the Max Error (|applied - actual|), the EI response will return an error
- all F32 (that is, “measurement”) data is in currently selected user engineering units

CMD2 = xxxx0101 (next point)			
U8	Recal Number	= 0x00-0xFF	must be same value used in “start FR”
U8	View/Perform	= 0x00-0xFF	must be same value used in “start FR”
U8	Cur Point	= 0x00-0xFF	must be same value returned from “start FR”/“next point”
U8	Num Points	= 0x00-0xFF	must be same value returned from “start FR”/“next point”

CMD2 = xxxx0110 (finish field recal)			
U8	Recal Number	= 0x00-0xFF	not used, references current field recal
U8	View/Perform	= 0x00-0xFF	not used, references current field recal
U8	Abort/Save	= 0x00-0xFF	0x00 = Abort recal, 0x01 = Save recal
U8	Disable/Enable	= 0x00-0xFF	0x00 = Disable recal, 0x01 = Enable recal

CMD2 = xxxx0111 (state field recal)			
U8	Recal Number	= 0x00-0xFF	not used, references current field recal
U8	View/Perform	= 0x00-0xFF	not used, references current field recal
U8	Get/Set	= 0x00-0xFF	0x00 = Get recal state, 0x01 = Set recal state
U8	Disable/Enable	= 0x00-0xFF	for Get: not used for Set: 0x00 = Disable recal, 0x01 = Enable recal

CMD_FIELD_RECAL (0x06) (continued)

Response Format – from Module to "Controller":

LEN	Length	= 0x??	length of DATA area
CMD1	Command1	= 0x06	echoed
CMD2	Command2	= 0x00-0xFF	echoed
STAT	Status	= 0x00-0xFF	general status, see status page

DATA			depends upon lower nibble of CMD2 as shown next

CMD2 = xxxx0000, xxxx0001			
U8	Status*	= 0x00-0xFF	individual status, see status page

* repeated in groups of 1 byte based the number of channels selected in CMD2 (for xxxx0000 only; valid for zero, offset and tare)

* for EVI zero (CMD3 = 0x60), six status bytes are returned corresponding to modes: MEAS_V, MEAS_I, MEAS_I2, SRCE_V, SRCE_I, and SINK_I

CMD2 = xxxx0010		
future		

CMD2 = xxxx0011 (start field recal)			
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
U8	Recal Number	= 0x00-0xFF	echoed
U8	View/Perform	= 0x00-0xFF	echoed
U8	Cur Point	= 0x00-0xFF	current point (x of n)
U8	Num Points	= 0x00-0xFF	# of points (n)
F32	Apply Point	=	recal value to apply
F32	Min Apply Point	=	min recal limit for this point
F32	Max Apply Point	=	max recal limit for this point
F32	Max Error	=	max error (|applied - actual|)

CMD2 = xxxx0100 (save point)			
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
U8	Recal Number	= 0x00-0xFF	echoed
U8	View/Perform	= 0x00-0xFF	echoed
U8	Cur Point	= 0x00-0xFF	echoed
U8	Num Points	= 0x00-0xFF	echoed
F32	Apply Point	=	echoed or previous value if “save point” failed
F32	Min Apply Point	=	min recal limit for this point
F32	Max Apply Point	=	max recal limit for this point
F32	Max Error	=	max error (|applied - actual|)

CMD2 = xxxx0101 (next point)			
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
U8	Recal Number	= 0x00-0xFF	echoed
U8	View/Perform	= 0x00-0xFF	echoed
U8	Cur Point	= 0x00-0xFF	the next point, which is now the current point
U8	Num Points	= 0x00-0xFF	echoed
F32	Apply Point	=	recal value to apply
F32	Min Apply Point	=	min recal limit for this point
F32	Max Apply Point	=	max recal limit for this point
F32	Max Error	=	max error (|applied - actual|)

CMD2 = xxxx0110 (finish field recal)			
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
U8	Recal Number	= 0x00-0xFF	echoed
U8	View/Perform	= 0x00-0xFF	echoed
U8	Abort/Save	= 0x00-0xFF	echoed
U8	Disable/Enable	= 0x00-0xFF	the current field recal state

CMD2 = xxxx0111 (state field recal)			
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
U8	Recal Number	= 0x00-0xFF	echoed
U8	View/Perform	= 0x00-0xFF	echoed
U8	Get/Set	= 0x00-0xFF	echoed
U8	Disable/Enable	= 0x00-0xFF	the current field recal state

Example	
To …	
	Controller--->EI:
	EI--->Controller:

CMD_GET_SET_RTCLOCK (0x07)

Command Format - from "Controller" to Module:

LEN	Length	= 0x??	length of DATA area		
CMD1	Command1	= 0x07			
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with channel:		
			xxx1 xxxx = spare		
			xx1x xxxx = spare		
			x1xx xxxx = spare		
			1xxx xxxx = set (get = 0)		
			Lower nibble not bit-encoded:		
			xxxx 0000 =	local time	
			xxxx 0001 =	coordinated universal time (UTC)	future
U16	Year*	=	self-explanatory		
U8	Month*	= 0x01-0x0C	| (1-12)		
U8	Day*	= 0x01-0x1F	| (1-31)		
U8	Hour*	= 0x00-0x17	| (0-23)		
U8	Minute*	= 0x00-0x3B	| (0-59)		
U8	Second*	= 0x00-0x3B	| (0-59)		
U8	DOW*	= 0x00	0 for now		
U8	Mode*	= 0x02	2 for now, 0=AM, 1=PM, 2=24HR		
S8	Offset*	= 0x00	0 for now, eventually offset from UTC		

* these parameters are not used (and need not be present) for get

Response Format – from Module to "Controller":

LEN	Length	= 0x0C	length of DATA area
CMD1	Command1	= 0x07	echoed
CMD2	Command2	= 0x00-0xFF	echoed
STAT	Status	= 0x00-0xFF	general status, see status page
U8	Status	= 0x00-0xFF	individual status, see status page
U8	Spare	= 0x00	spare to align on Word boundary
U16	Year	=	self-explanatory
U8	Month	= 0x01-0x0C	| (1-12)
U8	Day	= 0x01-0x1F	| (1-31)
U8	Hour	= 0x00-0x17	| (0-23)
U8	Minute	= 0x00-0x3B	| (0-59)
U8	Second	= 0x00-0x3B	| (0-59)
U8	DOW	= 0x00	0 for now
U8	Mode	= 0x02	2 for now, 0=AM, 1=PM, 2=24HR
S8	Offset	= 0x00	0 for now, eventually offset from UTC

Example	
To …	
	Controller--->EI:
	EI--->Controller:

CMD_GET_SET_FILTER (0x08)

Command Format - from "Controller" to Module:

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x08		
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with channel:	
			xxx1 xxxx = channel 1	
			EPI (Pressure): measure P1 pressure	
			EVI (Volt Amp): meas/sim volts	
			PS (M400): meas/sim HV volts	
			xx1x xxxx = channel 2	
			EPI (Pressure): measure P2 pressure	
			EVI (Volt Amp): meas/sim milliamps	
			PS (M400): meas Bus Vcc volts	
			x1xx xxxx = channel 3	
			ExI (that is, all): spare	
			PS (M400): meas Battery percent charge	
			1xxx xxxx = channel 4:	
			ExI (that is, all): measure internal temperature	
			Lower nibble not bit-encoded:	
			xxxx 0000 =	get current damp(s) for the specified channel(s)
			xxxx 0001 =	set specified damp(s) for the specified channel(s)
U8	State*	= 0x00-0xFF	0x00 = off, 0x01 = on, intended to be bit-encoded	
U8	Type*	= 0x00	0x00 = exponential damp, 0x01 = smart damp	
F32	Value*	=	for type 0: desired damp in seconds	

* repeated in groups of 6 bytes based upon the number of channels selected in CMD2 these parameters are not used (and need not be present) for get

Response Format – from Module to "Controller":

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x08	ehcoed	
CMD2	Command2	= 0x00-0xFF	echoed	
STAT	Status**	= 0x00-0xFF	general status, see status page	
U8	Status**	= 0x00-0xFF	individual status, see status page	
U8	Spare**	= 0x00	spare to align on Word boundary	
U8	State**	= 0x00-0xFF	the current filter state	
U8	Type**	= 0x00	the current filter type	
F32	Value**	=	for get:	
			• the current damp in seconds	
			for set:	
			• if specified value was valid, the specified value	
			• if specified value was invalid (too low/high), the current value is NOT changed and the corresponding status and damp limit (low/high) will returned	

U8	Data*	= 0x00-0xFF	parameter specified by lower nibble of CMD2
			notes:
			• for Network address, Data = 0x01 to 0xEF • for Module address, Data = 0x10 to 0x70 • for Baud rate index, Data = 0x00 to 0x08 o 0x00 = 19200 baud o

** repeated in groups of 8 bytes based upon the number of channels selected in CMD2

Example	
To …	
	Controller--->EI:
	EI--->Controller:

CMD_GET_SET_COMM (0x09)

Command Format - from "Controller" to Module:

LEN	Length	= 0x??	length of DATA area	
CMD1	Command1	= 0x09	echoed	
CMD2	Command2	= 0x00-0xFF	Upper nibble bit-encoded with attributes:	
			xxx1 xxxx = spare	
			xx1x xxxx = spare	
			x1xx xxxx = spare	
			1xxx xxxx = set (get = 0)	
			Lower nibble not bit-encoded:	
			xxxx 0000 = Network address	
			xxxx 0001 = Module address	
			xxxx 0010 = Baud rate index	
			xxxx 1111 = Offline Mode**	
U8	Data*	= 0x00-0xFF	parameter specified by lower nibble of CMD2	
			notes:	
			• for Network address, Data = 0x01 to 0xEF • for Module address, Data = 0x10 to 0x70 • for Baud rate index, Data = 0x00 to 0x08 o 0x00 = 19200 baud o 0x01 = 1200 baud - future o 0x02 = 2400 baud – future o 0x03 = 4800 baud – future o 0x04 = 9600 baud o 0x05 = 19200 baud o 0x06 = 38400 baud o 0x07 = 57600 baud o 0x08 = 115200 baud • for Offline Mode, Data = 0x00 to 0xFF** o 0x00 = Go online o 0x01-0xF0 = Go offline for x (1-240) minutes o 0xFF = Go offline until reset or power cycle	

* this parameter is not used (and need not be present) for get

** this command is immediate, a CMD_RESET (SOFT) will restore online functionality

NOTICE

This command is for experienced users.

NOTICE

A CMD_RESET (soft reboot) must be sent after completing Network address, Module address, and Baud rate changes to make them active.

CMD_GET_SET_COMM (0x09) (continued)

Response Format – from Module to "Controller":

LEN	Length	= 0x03	length of DATA area	
CMD1	Command1	= 0x09	echoed	
CMD2	Command2	= 0x00-0xFF	echoed	
STAT	Status	= 0x00-0xFF	general status, see status page	
U8	Status	= 0x00-0xFF	individual status, see status page	
U8	Spare	= 0x00	spare to align on Word boundary	
U8	Data	= 0x00-0xFF	the current value of the parameter	

Example	
To …	
	Controller--->EI:
	EI--->Controller:

NOTICE

This command is for experienced users.

NOTICE

A CMD_RESET (soft reboot) must be sent after completing comm. changes to make them active.

General Status

General Status (in Response Header):

[was the Command processed by the Subordinate?]

0x00 good

Miscellaneous: 0x01-0x0F

0x01 instrument busy, message discarded
0x02 message CRC invalid, message discarded
0x03 message incomplete after timeout, message discarded

Bad Header argument: 0x10-0x1F

0x10 command1 not supported or invalid
0x11 command2 not supported or invalid
0x12 command3 not supported or invalid
0x13 command1 not supported in current mode future
0x14 command2 not supported in current mode future
0x15 command3 not supported in current mode future

Bootloader-specific: 0xB0-0xBF

0xB0 command1 invalid in bootloader
0xB1 command2 invalid in bootloader
0xB2 command3 invalid in bootloader

Ramflash-specific: 0xC0-0xCF

0xC0 command1 invalid in ramflash
0xC1 command2 invalid in ramflash
0xC2 command3 invalid in ramflash

Production and/or Hardware failures: 0xF0-0xFF

0xF0 POST (power on self test) failed - general

0xF1 hardware missing/incomplete/failed

0xF2 main program not loaded, bootloader only

0xF3 memory map blank/not loaded

0xF4 memory map version/revision unsupported

0xF5 memory map class/type mismatch

0xF6 key fault detected

NOTICE

Always check the General Status.

- If the General Status is anything other than 0x00 (good), the Response payload will most likely be suppressed (except for the Extended Addressing). In the unlikely event the Payload is present, ignore it.
- Again, if the General Status is not 0x00 (good), the Subordinate could not process the Command.

Individual Status

Individual Status (in Response Data): [is the requested data valid?]

0x00 good

Miscellaneous: 0x01-0x0F

0x01 specified value invalid
0x02 memory/data location invalid
0x03 sensor not present or invalid
0x04 memory/data get/set failed
0x05 cmd1/2/3 not supported for this channel
0x06 payload arguments/data invalid
0x07 specified command is being processed
0x08 sensor not active in current mode
0x0F a general catch-all status

Calibration: 0x10-0x1F

0x10 cannot find cal data, primary meas too low
0x11 cannot find cal data, primary meas too high
0x12 cannot find cal data, secondary meas too low
0x13 cannot find cal data, secondary meas too high
0x14 calibration expired

Measurement: 0x20-0x2F

0x20 measurement soft under/over range
0x21 measurement hard under/over range
0x22 temperature soft under/over range
0x23 temperature hard under/over range

Simulation: 0x30-0x3F

0x30 simulation value too low
0x31 simulation value too high
0x32 simulation/output is at minimum value
0x33 simulation/output is at maximum value
0x34 simulation/output is under current (maybe open)
0x35 simulation/output is over current (maybe short)

Field Recalibration: 0x40-0x4F

0x40 field recal not allowed
0x41 too far from zero to zero
0x42 recal point outside valid range
0x43 recal point error beyond limit
0x44 general recal script error
0x45 general recal point library error
0x46 recal command out of sequence

Individual Status (in Response Data):

[is the requested data valid?]

SD Card/Data Logging: 0x50-0x5F

0x50 tbd
0x51 tbd
0x52 tbd
0x53 tbd
0x54 tbd
0x55 tbd
0x56 tbd
0x57 tbd
0x58 tbd
0x59 tbd
0x5A tbd

Power Delivery: 0x60-0x6F

0x60 no batteries installed
0x61 batteries too low for unit function
0x62 batteries nearing limit for unit function
0x63 USB power applied
0x64 sourcing function tripped (overcurrent)

Task Execution: 0x80-0x8F

0x80 specified task not supported/invalid
0x81 specified task not active
0x82 specified task active

NOTICE

Always check the General Status first, then check each of the Individual Statuses.

NOTICE

If a given Individual Status is anything other than 0x00 (good), the corresponding Response payload data will most likely be zero. In the unlikely event the corresponding Response payload data is not zero, ignore it.

Appendix A

Module Classes and Types

```
#define C_MEASUREMENT_SIMULATION    0x00    //Measurement/Simulation Class
#define T_EPI                       0       //Embedded Pressure Instrument Type
#define T_EVI                       1       //Embedded Volt Current Instrument Type
#define T_EIO                       2       //Embedded Digital IO Instrument Type
#define T_EAO                       3       //Embedded Analog Out Instrument Type
//
#define C_COMMUNICATIONS_BRIDGE     0x01    //Communications/Bridge Class
#define T_RS232_RS485               0       //RS232/RS485 Type
#define T_USB20                     1       //USB2.0 Type
//
#define C_REPOSITORY_DATALOGGING    0x02    //Repository/Data logging Class
#define T_REPOSITORY                0       //Repository Type
//
#define C_CONTROL_USER_INTERFACE    0x03    //Control/User Interface Class
#define T_CONTROL_M4xx              0       //Control Type
#define T_GRAPHICS_M4xx             1       //Graphics Type
//
#define C_POWER_SUPPLY              0x04    //Power Supply Class
#define T_M4xx                      0       //M4xx Type
#define T_VMA                       1       //VMA Type
```

Module Default Addresses

```
#define D_BROADCAST_ADDR            0x00
#define D_WILDCARD_ADDR             0xF0

#define D_MIN_NETW_ADDR             0x01        //minimum external network
address
#define D_MAX_NETW_ADDR             0xF0        //maximum external network
address
//
#define D_MIN_I2C_ADDR              0x10        //minimum I2C address
#define D_MAX_I2C_ADDR              0x70        //maximum I2C address
#define D_MODULE_ADDR_BSL           D_MAX_I2C_ADDR  //module address used by BSL if
                                                //EE value out of above range
//
#define D_BROADCAST_ADDR            0x00        //"broadcast" address
#define D_WILDCARD_ADDR             0xF0        //"unused" address
//
//                                              Communications/Bridge Class
#define D_MODULE_ADDR_COMM          0x28        //|  same address for all
#define D_BRIDGE_ADDR_COMM          D_WILDCARD_ADDR  //|  Types (RS232, USB, and so on)
#define D_NETWORK_ADDR_COMM         D_MODULE_ADDR_COMM  //|  within this Class
//
//                                              Control/User Interface Class
#define D_MODULE_ADDR_CONTROL       0x10        //Control
#define D_BRIDGE_ADDR_CONTROL       D_WILDCARD_ADDR  //|
#define D_NETWORK_ADDR_CONTROL      D_WILDCARD_ADDR  //|
//
#define D_MODULE_ADDR_GRAPHICS      0x11        //Graphics
#define D_BRIDGE_ADDR_GRAPHICS      D_WILDCARD_ADDR  //|
#define D_NETWORK_ADDR_GRAPHICS     D_WILDCARD_ADDR  //|
//
//                                              Power Supply Class
#define D_MODULE_ADDR_POWER         0x18        //|
#define D_BRIDGE_ADDR_POWER         D_WILDCARD_ADDR  //|
#define D_NETWORK_ADDR_POWER        D_WILDCARD_ADDR  //|
```

```
//
//                                                  Repository/Data logging Class
#define D_MODULE_ADDR_REPOSITORY    0x20            //Repository
#define D_BRIDGE_ADDR_REPOSITORY    D_WILDCARD_ADDR //|
#define D_NETWORK_ADDR_REPOSITORY   D_WILDCARD_ADDR //|
//
//                                                  Measurement/Simulation Class
#define D_MODULE_ADDR_EPI           0x40            //Embedded Pressure Instrument
#define D_BRIDGE_ADDR_EPI           D_WILDCARD_ADDR //|
#define D_NETWORK_ADDR_EPI          D_WILDCARD_ADDR //|
//
#define D_MODULE_ADDR_EVI           0x41            //Embedded Volt Current Instrument
#define D_BRIDGE_ADDR_EVI           D_WILDCARD_ADDR //|
#define D_NETWORK_ADDR_EVI          D_WILDCARD_ADDR //|
//
#define D_MODULE_ADDR_EIO           0x42            //Embedded Digital IO Instrument
#define D_BRIDGE_ADDR_EIO           D_WILDCARD_ADDR //|
#define D_NETWORK_ADDR_EIO          D_WILDCARD_ADDR //|
//
#define D_MODULE_ADDR_EAO           0x43            //Embedded Analog Out Instrument
#define D_BRIDGE_ADDR_EAO           D_WILDCARD_ADDR //|
#define D_NETWORK_ADDR_EAO          D_WILDCARD_ADDR //|
```

CRC16 Detail:

Normal (that is, not reflected) CRC-16-CCITT.

A CRC16 of "123456789" returns 0x31C3.

For a 18-byte message (as shown below in Red):

1. CRC16 bytes 1-10 of the header,
2. CRC16 bytes 13-18 of the payload,
3. insert the CRC16 into bytes 11 and 12, little-endian.

An example of CMD_GET_MEAS (internal temperature channel) from a PC to an RS232 module.

The Command is Red, the Response is Blue, the CRC16s are boxed.

Notice extended addressing is used.

```
TX: 80 01 00 03 28 04 80 00 00 00 D5 21 03 80 80 28 F0 2A

RX: 40 01 08 28 03 04 80 00 00 00 8A 40 00 01 02 00
RX: 91 7F 00 42 28 F0 2A 03 80 80
```

Message/Protocol Transmit and Receive Detail

RS232, USB20, I2C, and UART:

Steps in text:

1. Controller assembles the Command message,
2. Controller transmits the entire Command message to the EI,
3. EI processes command and assembles the Response message,
4. EI transmits the entire Response message to the Controller.

Steps in diagram:

Controller	EI
1.	Processing
2.	Command Header + Data>>>
3.	Processing
4.	<<<Response Header + Data

Note: *The Controller should wait >= 5mSec after receiving a Response before issuing another Command.*

Hardware and Firmware Communication Support

EPI (Embedded Pressure Instrument - Modular):

- EI commands: most
- General Interface: SPI, UART, I2C
- Production Interface: I2C
- TU1 = at 19,200-115,200 baud, at 8MHz, no pause necessary
- TS1 = TBD
    - the Master must wait >= 5 mSec (from completion of Subordinate response) before issuing another command (actually, this time may be as low as >= 1 mSec – try at own risk)

Note: *SPI/UART autodetect TBD, UART autobaud detect TBD*

EVI (Embedded Volt Current Instrument - Modular):

- EI commands: most
- General Interface: SPI, UART, I2C
- Production Interface: I2C
- TU1 = TBD
- TS1 = TBD

Note: *SPI/UART autodetect TBD, UART autobaud detect TBD*

RS232485 and USB Comm. (Modular):

- EI commands: 0x00, 0x02-0x04, 0x60-0x62
- General Interface: Specific to type of comm. module
- Production Interface: I2C
- TU1 = n/a
- TS1 = n/a