



Meriam Serial Protocol Implementation Guide

for

**M330 Embedded Pressure Instrument
M1500 Digital Pressure Transmitter**

Table of Contents:

Using the M330 Development Interface (M330 products only) 3

Preface 4

Overview 4

 Message Structure 5

 Communication Protocol 5

 Terminology 5

 Command Format 6

 Data Types 6

 Response Format 7

 Module Classes, Types & Default Addresses 8

Commands/Responses

 CMD_RESET (0x00) perform various resets 9

 CMD_GET_SET_INFO (0x02) get/set basic and calibration info 10

 CMD_GET_SET_UNITS (0x03) get/set CMS* engineering units 12

 CMD_GET_MEAS (0x04) get CMS* measurements 15

 CMD_MEAS_SIM_MODE (0x05) not released

 CMD_FIELD_RECAL (0x06) perform field recalibrations (incl. Zero) 17

 CMD_GET_SET_RTCLOCK (0x07) not released

 * CMS – calibrated measurement/simulation

Status List

 General 21

 Individual 22

Appendix A

 CRC16 Detail 23

 Message/Protocol Transmit and Receive Detail 24

Appendix B

 EIProtocol.h file information 25

Using the M330 Development Interface

The Meriam M330 EPI is an accurate, self-contained, fully compensated, digital pressure measurement instrument designed to be tightly integrated into a larger system. Digital communication with the M330 EPI is accommodated via I²C or UART (both standard) or SPI (available order option). Meriam Serial Protocol (MSP) is used to communicate with the M330.



M330 Embedded Pressure Instrument

The M330 Development Kit (pn Z9A450) allows the M330 to connect directly to a host PC (using the included USB cable) for evaluation or software development activities. Customers will need the M330 Development Kit, a PC and expertise with suitable development software (C/C++, Visual Basic) to design M330 communications software for their system. The MSP Implementation Guide provides the protocol command strings for the M330. The guide is included with each M330 EPI and M330 Development Kit (see the Product CD) or can be downloaded at www.meriam.com.

M330 Development Kit & Instructions

The M330 Development Kit includes the following components:

Part Number	Description
Z9A449	M330 Development Interface (hardware)
Z9P342	USB type A (male) by USB type B (male)
Z9A000003PN06	Product CD including MSP Implementation Guide, Modbus RTU Implementation Guide, Source Code Samples, USB Drivers

To develop communication software for the M330:

1. Install USB Drivers on the host PC (drivers and instructions are found on the supplied Product CD)
2. Connect a M330 EPI unit to the M330 Development Interface module as shown below.
3. Use the Kit's USB cable to connect the assembly to the host PC.
4. Use the Meriam Serial Protocol Implementation Guide, as desired, to develop the communication software for the application.



M330 Embedded Pressure Instrument

M330 Development Interface

USB Cable

Contact Meriam Technical Support (sales@meriam.com) with questions that arise.

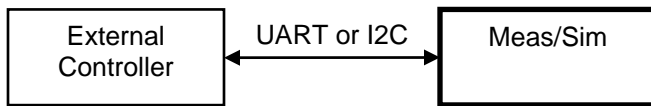
Preface:

Meriam’s new modular-based products are comprised of the following “Classes” and “Types” of “Modules” which can be used alone or in combination to make a finished good.

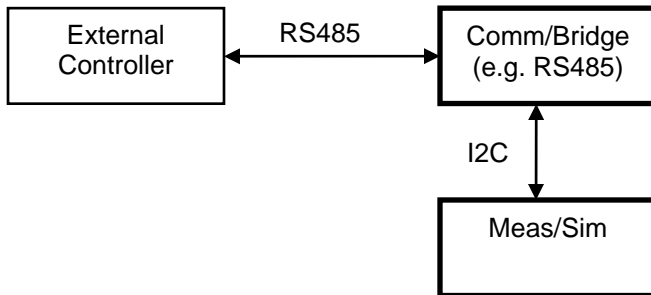
<u>Class</u>	<u>Type</u>
Measurement/Simulation	EPI (pressure), EVI (volt, mA)
Communications/Bridge	RS-232/RS-485, USB20
Repository/Data logging	Repository

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

Embedded Instrument: example - EPI



Short-Stack: example – M1500



Overview:

This document describes the message structure and communication protocol between a Controller and an Embedded Instrument or Short-Stack.

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

Of course, not all commands are supported on all modules.

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.

Communication Protocol:

There are two messages types:

- Command
- Response

The Command message is sent from the Controller (the Master) to the Embedded Instrument or Short-Stack (the Slave). 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 Slave

- The Master transmits a Command Message and receives a Response Message.

Master = the side that initiates communication

- The Master is not typically able to receive an unsolicited response message.

Slave = the side the responds to communication

- The Slave is almost always ready to receive an unsolicited command message.
- The Slave 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 Slave composes a Response Message

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

- Source = 1 byte: typically Module address specified in the Header
- Destination = 1 byte: typically Module address specified in the Header

Extended Message Addressing = addressing required to externally access (e.g. PC app, etc.) 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

Command Format - from “Controller” to Embedded Instrument or Short-Stack:

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	Srcce 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) bit: lxxx xxxx = SRC = for this command
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	(i.e. 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 command originator and the complete address of the intended command recipient
n+1	SNET	SrcceNetwork	= 0x00-0xFF	source network address
n+2	SBRI	SrcceBridge	= 0x10-0x70	source bridge address
n+3	SMOD	SrcceModule	= 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

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 Embedded Instrument or Short-Stack 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	Srcce 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	(i.e. 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 (i.e. intended command recipient) and the complete address of the response recipient (i.e. command originator)				
n+1	SNET	SrcceNetwork	= 0x00-0xFF	source network address
n+2	SBRI	SrcceBridge	= 0x10-0x70	source bridge address
n+3	SMOD	SrcceModule	= 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

Module Classes, Types and Default Addresses

```

#define C_MEASUREMENT_SIMULATION 0x00 //Measurement/Simulation Class
#define T_EPI 0 //Embedded Pressure Instrument Type
#define T_EVI 1 //Embedded Volt Cur 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 D_BROADCAST_ADDR 0x00
#define D_WILDCARD_ADDR 0xF0

//Communications/Bridge Class
#define D_MODULE_ADDR_COMM 0x28 //| same address for all
#define D_BRIDGE_ADDR_COMM D_WILDCARD_ADDR //| Types (RS232, USB, etc.)
#define D_NETWORK_ADDR_COMM D_MODULE_ADDR_COMM //| within this Class
//
//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 Cur Instrument
#define D_BRIDGE_ADDR_EVI D_WILDCARD_ADDR //|
#define D_NETWORK_ADDR_EVI D_WILDCARD_ADDR //|

```

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, etc.) as shown on the preceding pages.

A support file, EIProtocol.h, contains defines (like the ones directly above) and data structures/unions that support the following commands.

Note: for information on where it find the latest EIProtocol.h file, see Appendix B

CMD_RESET (0x00)

Command Format - from "Controller" to Embedded Instrument:

LEN	Length	= 0x00	length of DATA area
CMD1	Command1	= 0x00	
CMD2	Command2	= 0x00-0xFF	Types of reset: 0x00 = complete reset, soft reboot

Response Format - from Embedded Instrument to "Controller":

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

Example:

```
To ...  
Controller--->EI:  
  
EI--->Controller:
```

CMD_GET_SET_INFO (0x02)**Command Format - from "Controller" to Embedded Instrument:**

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
			lxxx xxxx = set (get = 0)	set is future
			Lower nibble not bit-encoded:	
			xxxx 0000 = normal (EI) list	
CMD3	Command3	= 0x00-0xFF	reference number	

U8	Data Bytes	=	data for a set (write)	

Response Format - from Embedded Instrument 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: SNs, class, type, addresses, etc.

0x11 = sensor 1 measurement info: sensor type, limits, accuracy, etc.

0x21 = sensor 2 measurement info: same as 0x11

Example structures (always refer to EIProtocol.h):

```

struct
{
    Byte  stat;                //individual status, see status page
    Byte  bRunningCode;       //currently running code (BOOT, RAM, FW)
    char  szStackSN[12];
    char  szModuleSN[12];
    Byte  bClass;
    Byte  bType;
    Byte  bHardwareRev;
    Byte  bMemMapRev;
    char  szFirmwareRev[8];
    Byte  bNetworkAddr;       //the current address
    Byte  bBridgeAddr;        //|
    Byte  bModuleAddr;        //|
    Byte  bSpare;

} r00;                        //response r00, 42 bytes

struct
{
    Byte  stat;                //individual status, see status page
    Byte  pad;                 //spare to align on Word boundary
    Byte  bSensorType;
    Byte  bNativeUnits;       //native unit of sensor (PSI for pressure)
    Byte  bSplashUnits;       //"sold as" engineering unit (e.g. 200 inW20C)
    Byte  bSpare;
    float fLSL;               //"sold as" lower sensor unit in splash units
    float fUSL;               //"sold as" upper sensor limit in splash units
    Byte  bAccyType;
    Byte  bSpare2;
    Byte  bAccyData[16];

} r11;                        //response r11, 32 bytes

```

CMD_GET_SET_UNITS (0x03)**Command Format - from "Controller" to Embedded Instrument:**

```

LEN      Length      = 0x??          length of DATA area
CMD1     Command1    = 0x03
CMD2     Command2    = 0x00-0xFF

Upper nibble bit-encoded with channel:
xxxx1 xxxx = channel 1
      EPI (Pressure):  measure P1 pressure
      EVI (Volt Amp):  meas/sim volts
xx1x xxxx = channel 2
      EPI (Pressure):  measure P2 pressure
      EVI (Volt Amp):  meas/sim milliamps
x1xx xxxx = channel 3
      ExI (i.e. all):  spare
1xxx xxxx = channel 4:
      ExI (i.e. 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 (i.e. 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 Embedded Instrument 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 req'd
U8	Spare**	= 0x00	spare to align to Word boundary
U8	Unit Text**	= 0x00-0xFF	short units text string (e.g. "inW20C")
U8	**	= 0x00-0xFF	up to 6 characters + NULL, left-justified
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:

0	mA DC
1	V DC

MAP Internal Temperature Channel Units:

0	°F
1	°C
2	K
3	°R

CMD_GET_MEAS (0x04)**Command Format - from "Controller" to Embedded Instrument:**

```

LEN      Length      = 0x00      length of DATA area
CMD1     Command1     = 0x04
CMD2     Command2     = 0x00-0xFF

Upper nibble bit-encoded with channel:
xxxx1 xxxx = channel 1
      EPI (Pressure):  measure P1 pressure
      EVI (Volt Amp):  meas/sim volts
xx1x xxxx = channel 2
      EPI (Pressure):  measure P2 pressure
      EVI (Volt Amp):  meas/sim milliamps
x1xx xxxx = channel 3
      ExI (i.e. all):  spare
1xxx xxxx = channel 4:
      ExI (i.e. 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 = same as 0010, but also gets scaled measurement
          integers (0-65535) for specified channel(s)

note:
      this command is mode-dependent; it will get/set/read
      the units for the active mode (i.e. meas or sim)

```

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

CMD_GET_MEAS (0x04) (continued)**Response Format - from Embedded Instrument to "Controller":**

LEN	Length	= 0x??	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
<u>CMD2 = xxxx0000, xxxx0001</u>			
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
			notes:
			the above xODs are signed Bytes
			the above xODs are actual/meas-specific for the unit
			if either ROD is < 0, scientific notation is req'd
U8	Spare	= 0x00	spare to align Float on Word boundary
F32	Measurement	=	measurement data, 32-bit float in little-endian
<u>CMD2 = xxxx0010</u>			
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
<u>CMD2 = xxxx0011</u>			
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
U16	Scaled	=	scaled meas data, 16-bit unsigned int in little-endian

Example:

```
To ...
    Controller--->EI:

    EI--->Controller:
```


CMD_FIELD_RECAL (0x06)**Command Format - from "Controller" to Embedded Instrument:**

```
LEN      Length      = 0x??      length of DATA area
CMD1     Command1     = 0x06
CMD2     Command2     = 0x00-0xFF  Upper nibble bit-encoded with channel:
                      xxxx 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 (i.e. all):  spare
                      1xxx xxxx = channel 4:
                      ExI (i.e. all):  measure internal temperature
note: only one channel supported at a time,
      except for zero

Lower nibble not bit-encoded:
xxxx 0000 = zero the specified channel(s)
xxxx 0001 = restore factory defaults
xxxx 0010 = get supported field recals      - future
xxxx 0011 = start field recal
xxxx 0100 = save point
xxxx 0101 = next point
xxxx 0110 = finish field recal
xxxx 0111 = state field recal

xxxx 1111 = internal temperature          - future

----
DATA                                     depends upon lower nibble of CMD2 as shown next

CMD2 = xxxx0000
NO DATA (just command header)

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

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 must be same value used in "start FR"
U8 View/Perform = 0x00-0xFF must be same value used in "start FR"
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 Embedded Instrument 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)

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:
```

Status

General Status (in Response Header):

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

0x14 command2 not supported in current mode

0x15 command3 not supported in current mode

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

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

Note: this page is continually being expanded, latest constants in EIProtocol.h

Status (continued)

Individual Status (in Response Data):

0x00 good

Miscellaneous: 0x01-0x0F

0x01 engineering unit 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
0x0F a general catch-all status

Calibration: 0x10-0x1F

0x14 calibration expired

Measurement: 0x20-0x2F

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

Simulation: 0x30-0x3F

0x30 simulation value too low
0x31 simulation value too high

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

Note: this page is continually being expanded, latest constants in EIProtocol.h

Appendix A

CRC16 Detail:

Normal (i.e. 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:

- | | <u>Controller</u> | <u>EI</u> |
|----|---------------------------|-----------|
| 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.

Message/Protocol Transmit and Receive Detail (continued):SPI: (for *advanced* users only)

Steps in text:

1. Controller assembles the Command message,
2. Controller clocks-out the Command message Header to the EI,
EI clocks 0x00s back
3. Controller waits TS1* (while the EI parses length from message header and sets up DMA to receive balance of message),
4. Controller clocks-out the Command message Data to the EI,
EI clocks 0x01s back
5. Controller waits TS1* (while the EI completes command receipt and loads 0x02 into transmit register), then Controller begins clocking 0x03s**
EI clocks 0x02s back while BUSY (processing the command and assembling the Response message)
6. Controller clocks-out 12 (Response message Header size) 0x03s,
EI clocks the Response message Header back
7. Controller waits TS1* (while it parses length from message header and sets up DMA to receive balance of message),
8. Controller clocks-out Length (from Response message Header) 0x04s.
EI clocks the Response message Data back

Steps in diagram:

<u>Controller</u>	<u>EI</u>
1.	Processing
2.	Command Header>>> <<<all 0x00s
3.	Pause
4.	Command Data>>> <<<all 0x01s
5.	Busy to Ready** <<<all 0x02s
6.	12 0x03s>>> <<<Response Header
7.	Pause
8.	Length 0x04s>>> <<<Response Data

*TS1 – 5mSec

**BUSY indication (done in protocol, no hardware line). The Controller can poll the EI by clocking-out 0x03s. If the Controller receives 0x02s, the EI is BUSY – as soon as the Controller receives Response Preamble1 (0x40), the Controller must clock-out another 11 0x03s to complete receipt of the Response Header. While the EI is BUSY, the Controller can deselect it, go perform another task, and reselect the EI and check the status (i.e. send 0x03s). So, the Controller does NOT have to wait (i.e. keep the EI selected and keep clocking 0x03s) while the EI is BUSY.

Appendix B

EIProtocol.h file:

The latest EIProtocol.h file for use in programming is available at www.meriam.com