The Compact Ethernet Communication (CEC) Protocol

Version 1.1

G. Saewert, B. Kramper

11/15/05

This document defines an Ethernet communication protocol for use between an ACNET front end and an embedded hardware controller. The intent of this protocol is to extend ACNET services to an embedded controller that does not necessarily have the resources to be an ACNET front end. The protocol is compact in that it defines a minimum of what is necessary to control a hardware system via ACNET. The protocol, being minimalistic, enables hardware systems containing microcontrollers, small microprocessor core modules or DSPs to be controllable by ACNET provided they are Ethernet enabled. This protocol is extendable and contains features to debug communication messages.

It is assumed the client is the ACNET front end and the server is the Ethernet controller. Front ends can be any valid ACNET front end including, but not exclusive to, VME processors running MOOC on VxWorks; an open access client written in Java running on one of the controls department data engines; a computer running LabView with drivers to service ACNET requests; or an Internet Rack Monitor.

1.0 Client/Server Communication Rules

1. UDP will be used as the communication protocol over Ethernet.
2. The server will not initiate communication. The server will only transmit a datagram in response to a request first sent to the server by the client.
3. The server will always send an acknowledgement message back to the client in response to every request received.
4. Both client and server will transmit data following network byte order convention, namely "big endian".

2.0 UDP Datagram Message Structure

Clients send messages to the server always with a command to do something. A command will be either to set a parameter to some value or to read back parameter values. This protocol requires that servers always reply back to the client with a reply message. Replies sent in response to commands to set a parameter value will be simply an acknowledgement of the command. It will echo back the command message exactly as it received it and will include an error code. Replies to the client returning read back data will also echo back the command message with an error code and will include the requested data.

Table 2.0 illustrates the UDP datagram message structure typical of all messages. The message header appears first followed by data, if there is any. All messages sent by client and server will always contain a message header, but not all messages will contain data. Command messages to set a value will include the data value to set, but those to read data back will not include any data. Replies to the client will always contain data.

Each application will have defined for it the data that will be exchanged between client and server. This data will be arranged into arrays—one array for each device type. There are four fundamentally essential device types identified in this document: readings, settings, control bits and status. This document assumes data will be transmitted in arrays of only these types. The information in the message header makes reference to one, and only one, of these defined arrays in any given transmission.

The following sections explain each message header element.

Table 2.0. *Datagram Message Structure.* `Short` *refers to a 16-bit signed integer.*

Message Element		Description
Message Header	`short` *byte_length*	Byte length of the message, including the *byte_length* field and data.
	`short` *message_type*	A code identifying the type of message sent.
	`short` *initial_element*	The index of the first device element of the device type indicated in the message type.
	`short` *element_qty*	Quantity of data elements being referenced in the message request.
	`short` *error_code*	Success code returned by the server.
Data	`Data`	Settings, control bit mask, digital status or read back of analog parameter values.

2.1 *Byte_length*

This element indicates the number of bytes of the entire message sent including the *byte_length* element.

2.2 *Message_type*

Implicit in the message type code is both the command of what the server is to do and which device type data array is involved. Table 2.2 lists the messages codes for reading and setting the essential readable and settable types of devices.

When returning the acknowledgement to the front end, the server echoes back the same message type code as was sent in the client request. Table 2.2 lists the valid message type code values. (These messages are a subset of existing ACNET property indexes, although the code values are different.)

Table 2.2. *Message Types*.

Type Code	Description
0	Read request (analog values).
1	Read setting request (analog setting values).
2	Read status request.
3	Set a setting value (analog settings).
4	Set a control bit.

2.3 *Initial_element*

This is an index number pointing to one of the array elements of the defined device type data array indicated by the message type. The client may request to read back the parameter values of any number of the possible readable devices. If the request is to read back all the devices, the *initial_element* will have a value 0. If the client is requesting to read back a subset, then the *initial_element* is the index of the first element of interest in the data array. When the client is requesting to set a parameter's value, the initial element will be the index of the array element containing the data to be set.

2.4 *Element_qty*

This number indicates the quantity of elements being referred to by the message type. When setting parameters, the quantity will always be 1. To read back parameters the quantity will be either the total number of possible readable devices, or it will be a quantity to be included in a subset. Subsets of elements will always be a group of contiguous elements in the array of data of the type referred to by the message type.

2.5 *Error_code*

This element will be used in message replies that the server sends back to the client. Following common convention, a value of zero will indicate no error. A negative value will mean that there is some condition that should be interpreted as an error. Positive values will be informative only, like "pending", etc. Each application can utilize additional codes that are more specific, but defined codes are listed in Table 2.5. It is recommended not to redefine the code values already defined in Table 2.5.

Table 2.5. *Error Codes*.

Error Code	Meaning
1	Action of last request is pending. The action will be initiated after some other active process is complete.
0	Successful execution of last sent request.
-1	*Message_type* value was invalid.
-2	*initial_element* value was invalid.
-3	*element_qty* value was invalid.
-4	Setting value was out of range.
-5	*ftp_freq* value is too high to be supported.

2.6 *Data*

This is an array of data of the type indicated by the message type code. One message will contain data of only one device type. When the client is setting parameters, the data following the message header is the one setting value for the parameter to be changed. In a read back command, the device quantity message element will indicate the number of data values for the client to send, and the server will return an array of device data readings containing the number of which is given in device quantity message field.

Control bits are stored in arrays of words, one word for each device such as a power supply, for example. On a command to set the control bit of a particular device, the data value will be a mask that singles out a specific bit in the word. The initial device will indicate which of all the defined power supply devices is to have a bit changed.

Data type and structure is to be agreed upon between client and server depending on the application. Not all data will necessarily be 16-bit integers. Analog values may need to be understood as unsigned, for example. New message type codes could be defined that refer to aggregate types of devices that contain numerous elements per device. For any given type of device, nonetheless, the data will be in array format.

3.0 Implementation

This section describes the format for each message type exchanged between client and server listed in Table 2.2. In this section there is a table for both the client's request and the server's reply that show message header values that are valid for each of the five message types. Applications will not necessarily need to provide code to handle all five messages. If an application has no control bits to be set, software code need not be implemented to deal with control bit message type code 4, for example.

Steps to implement this protocol:

(1) Determine the data that will be communicated over Ethernet.
(2) Arrange this data into arrays: settings, readings, control and status.
(3) Determine desired ACNET parameters and construct them from the data arrays.
(4) Determine ACNET scale factors for settings and readings.

This section lists tables of message header values for each of the possible message types listed in Table 2.2. The following syntax is used in the tables:

- The message element value fields in which specific values are known contain the known numerical value.
- Fields contain the word "value" that will contain any 16-bit value.
- "MAX_ELEMENTS" refers to the number of elements in the data array for the particular message type.
- "data" indicates an array of data whose size depends on the specific request and application.
- "n/a" indicates an element value that is not used for that particular message, and in that case any numerical value will simply serve as a placeholder in the message header.

3.1 Read Request – Message Type 0

The client will issue a message of type "read request" when requesting read back of analog parameters. The client can request either the read back of all the readable analog devices or a subset.

Table 3.2a. *Client "Read Request" Message.*

Message element	Message Element Value	Comments
byte_length	10	
message_type	0	Read request.
initial_element	0 – (MAX_ELEMENTS - 1)	
element_qty	1 – MAX_ELEMENTS	
error_code	n/a	

Table 3.2b. *Server "Read Reply" Message.*

Message element	Message Element Value	Comments
byte_length	value	Depends on the amount of data returned.
message_type	0	Value echoed back.
initial_element	0 – (MAX_ELEMENTS - 1)	Value echoed back.
element_qty	1 – MAX_ELEMENTS	Value echoed back.
error_code	value	0 if no error, etc.
data	data	

3.2 Read Setting Request – Message Type 1

The client will issue a message of type "read setting request" when requesting to read back the analog setting values. The client can request either the read back of all the device settings or a subset.

Table 3.3a. *Client "Read Setting Request" Message.*

Message element	Message Element Value	Comments
byte_length	10	
message_type	1	Read settings.
initial_element	0 – (MAX_ELEMENTS - 1)	
element_qty	1 – MAX_ELEMENTS	
error_code	n/a	

Table 3.3b. *Server "Read Setting Reply" Message.*

Message element	Message Element Value	Comments
byte_length	value	Depends on the amount of data returned.
message_type	1	Value echoed back.
initial_element	0 – (MAX_ELEMENTS - 1)	Value echoed back.
element_qty	1 – MAX_ELEMENTS	Value echoed back.
error_code	value	0 if no error, etc.
data	data	

3.3 Read Status Request – Message Type 2

The client will issue a message to type "read status" when requesting to read back digital status. A status word exists for each device such as a power supply. The client can request either the read back of status for all existing devices or a subset.

Table 3.4a. *Client "Read Status Request" Message.*

Message element	Message Element Value	Comments
byte_length	10	
message_type	2	Read status.
initial_element	0 – (MAX_ELEMENTS - 1)	
element_qty	1 – MAX_ELEMENTS	
error_code	n/a	

Table 3.4b. *Server "Read Status Reply" Message.*

Message element	Message Element Value	Comments
byte_length	value	Depends on the amount of data returned.
message_type	2	Value echoed back.
initial_element	0 – (MAX_ELEMENTS - 1)	Value echoed back.
element_qty	1 – MAX_ELEMENTS	Value echoed back.
error_code	value	0 if no error, etc.
data	data	

3.4 Set a Setting Request – Message Type 3

The client will issue a message of type "set setting" when requesting to set the value of an analog settable parameter. The client will issue one separate request each parameter desired to be set.

Table 3.5a. *Client "Set Setting Value Request" Message.*

Message element	Message Element Value	Comments
byte_length	12	One value sent for *data*.
message_type	3	Set a setting.
initial_element	0 – (MAX_ELEMENTS - 1)	
element_qty	1	
error_code	n/a	
data	One value	Desired analog value.

Table 3.5b. *Server" Set Setting Value Reply" Message.*

Message element	Message Element Value	Comments
byte_length	12	One value sent for *data*.
message_type	3	Value echoed back.
initial_element	0 – (MAX_ELEMENTS - 1)	Value echoed back.
element_qty	1	Value echoed back.
error_code	value	0 if no error, etc.
data	One value	Value echoed back.

3.5 Set a Control Bit Request – Message Type 4

There is some flexibility here. The client will issue a message to "set a control bit" when requesting to set a single control bit. In the desire to keep functionality close to that of ACNET, there is nothing preventing the client from asserting more than one bit at once. This document does not prevent the implementation of more than one bit being set from one command.

Table 3.6a. *Client "Set Control Bit Request" Message.*

Message element	Message Element Value	Comments
byte_length	12	One value sent for *data*.
message_type	4	Set a control bit.
initial_element	0 – (MAX_ELEMENTS - 1)	
element_qty	1	
error_code	n/a	
data	One value	Desired control value.

Table 3.6b. *Server "Set Control Bit Reply" Message.*

Message element	Message Element Value	Comments
byte_length	12	One value sent for *data*.
message_type	4	Value echoed back.
initial_element	0 – (MAX_ELEMENTS - 1)	Value echoed back.
element_qty	1	Value echoed back.
error_code	value	0 if no error, etc.
data	One value	Value echoed back.

3.6 Extending the Protocol

This document has assumed data will be transmitted between client and server in only one of four types of data arrays. In addition, nothing has been mentioned about handling fast time plots. There is nothing preventing the communication of other data in structures other that these four arrays or handling fast time plot data. These can be accommodated by extending Table 2.2 to include additional device types and defining data structures to contain the desired data.

A. Revision History

10/22/04	Originated.
1/26/05	Wording changed to clarify device types and that aggregate structures will not be defined in this document.
1/28/05	Major changes. Renumbered sections. Creation of section 3 Implementation.
3/17/05	Changed title. Changed rule 4. in section 1.0.
4/8/05	Changed the status of this document from draft to version 1.0.
4/14/05	Removed device_type from message elements; moved error_code to the location after device quantity. Reworded the document to reflect the elimination of device_type.
4/15/05	Removed sending data on every client command message, only on those that make sense.
11/15/05	Gave this protocol a name, CEC. Clarified wording in section 3.