Delta Motion Control Protocol (DMCP)
User Guide

Why Another Protocol?

The RMC supports many application protocols; most were designed by other automation companies. However, when Delta Computer Systems, Inc. needed to choose a protocol to use for its own software, such as the RMCENET ActiveX Control, to communicate with the RMC, it found that none of these protocols satisfied this simple list of requirements:

1. It must have binary, little-Endian, word-aligned encoding.
2. It must have low data and processing overhead.
3. It must be flexible enough to allow addressing of a minimum of 65,536 16-bit registers.
4. It must be non-connection based to support both UDP and TCP operating modes.

Therefore, Delta designed yet another application protocol, called Delta Motion Control Protocol (DMCP).

DMCP versus Modbus/TCP

Users who want to control or monitor the RMC through TCP or UDP must select one of the protocols the RMC supports. Delta recommends two of these protocols: DMCP and Modbus/TCP. All other protocols supported by the RMC are either difficult to get information on, complicated, or inefficient. We will now compare DMCP and Modbus/TCP.

DMCP was designed by Delta Computer Systems for its own products and is not implemented by other manufacturers. Modbus/TCP is an open protocol maintained by Schneider Electric and implemented by many manufacturers; this is the main advantage of Modbus/TCP.

The following chart compares the technical specifications of these two protocols:

Characteristics	**DMCP**	**Modbus/TCP**
Transport Protocols	TCP, UDP	TCP
Byte Order	Big or Little Endian	Big Endian (Motorola)
Data Alignment	16-bit	8-bit
Static Header Size	7 bytes	8 bytes
Maximum 16-bit Registers per Packet	TCP: 2048 UDP: 512	Read: 125 Write: 100

Because of the above statistics, DMCP can be significantly faster than Modbus/TCP, as demonstrated by the following benchmark data:

Benchmarks	DMCP	Modbus/TCP
Read 125 words	TCP: 6.27 ms UDP: 5.21 ms	6.46 ms
Read 512 words	TCP: 12.0 ms UDP: 10.4 ms	31.2 ms (5 packets)
Read 2048 words	TCP: 35.9 ms UDP: 42.4 ms (4 packets)	109 ms (17 packets)

Although DMCP is up to three times faster for large packets, it is worth noting that the majority of transfers for most applications are to the status and command areas, which are both under 125 words. On those transfers the speed advantage is smallest. These statistics were taken from a Windows application requesting data from an RMC.

From the above information, the selection criteria can be simplified to the following:

- Choose Modbus/TCP if you want to be able to reuse your code with non-Delta Ethernet products in the future.
- Choose DMCP if you need the additional performance, especially sending large amounts of data.

The rest of this document is dedicated to defining DMCP. For more information on Modbus/TCP, see Schneider Electric's web site at www.modicon.com.

DMCP Characteristics

DMCP satisfies the above requirements, which restated and slightly appended, comes to this basic list of characteristics:

- DMCP servers will listen on registered TCP and/or UDP ports 1324. The RMC listens on both. It also listens on TCP and UDP private ports 50000 because this was the port used before we had a registered port. This private port is being phased out.

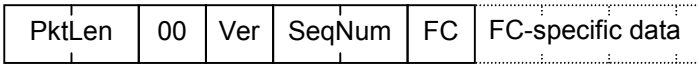
  NOTE: Prior to RMC ENET firmware 20001108, the RMC only listened on the private ports. However, these ports are being phased out, so we recommend updating to 20001108 or newer firmware rather than using the private ports.

- All values are binary encoded. Values larger than a byte will be word-aligned and be in little-Endian (LSB first) format. The exception to this is that the data contents of reads and writes can be issued in either big- or little-Endian, depending on the function used.
- No checksum is included in this application layer protocol since it this data is already checked in the TCP, UDP (optionally), and Ethernet protocols.
- No source and destination node addresses will be added to the protocol, as the source and destination IP addresses and port numbers are already available in the IP, TCP, and UDP layers.

DMCP Format

DMCP is a simple request/response protocol. Once the client has an open TCP or UDP connection, the client sends a request packet and the server responds with a packet. Of course, UDP has no communication connection, but some API's, including BSD sockets, allow opening a virtual connection to avoid entering the IP address in every function call. The client may send several requests—the server must respond in the order received—but it is intended that only a few requests are made at a time, so the server can be designed accordingly.

DMCP request packets have this simple format:



DMCP response packets have the exact same format as the request except that function codes will have response bits set (see below) and the FC-specific data will be different.

NOTE: Prior to RMC ENET firmware 20000420, the RMC used version 1 of the packet format, which is not documented here. We recommend instead upgrading to 20001108 or newer firmware.

The above fields are each described below:

- **PktLen**

  This field gives the number of *bytes* in the entire packet excluding only the 2-byte PktLen field. The valid range for this field is 5 to 1034 when using UDP and 5 to 4106 when using TCP.

- **Ver**

  This gives the packet format version. Currently, this should be 02.

- **SeqNum**

  The sequence number is echoed in the response from a server. The client need not use this field, but when the UDP transport is used, it is highly recommended that the sequence number be used to ensure that the responses received match the requests, since packets can arrive out of order.

- **FC (Function Code)**

  This single-byte field identifies the operation that will be made. The rest of this document describes function codes and their usage. Function codes will always be displayed in hexadecimal to simplify this document. Valid function code requests will always fall between 00 and 1F. Response function codes will be fall into two categories. Success response function codes are equal to the request function code plus 80 (i.e. the range is 80 to 9F). Failure response function codes are equal to the request function code plus 40 (i.e. the range is 40 to 5F).

- **FC-specific data**

  Many function codes require additional data in the request and/or response packets.

UDP Considerations

DMCP is designed to work over either TCP or UDP. However, because UDP does not support retrying packets and does not guarantee that packets will arrive in the order sent, a couple of additional issues should be considered. However, if used correctly, UDP will give higher performance than TCP and gives the user better control over timeouts, so advanced users may want to do the following:

- If IP packets are routed between networks, as opposed to be limited to a single network, they can arrive at the target out of order from when they are sent. TCP sequences the data before passing it to the application data, but UDP does not. Therefore, for UDP you *must* use the SeqNum field in the DMCP header to match responses with requests. Typically you will increment the SeqNum field in each new request.
- If a packet is lost, then eventually the client must decide to retry the packet. The client should use the DMCP response packet as the positive acknowledgement and a timeout as a negative acknowledgement. Therefore, the client must be designed such that it handles the case in which the response from the RMC is lost or delayed and that response comes back after the retry is sent. Therefore, two replies would come in a row. The client must discard the second unexpected reply.

Function Code List

Four function codes are currently defined for use outside Delta:

0x10	Read Device Registers (little Endian)
0x11	Write Device Registers (little Endian)
0x12	Read Device Registers (big Endian)
0x13	Write Device Registers (big Endian)

These function codes will be described below in numerical order.

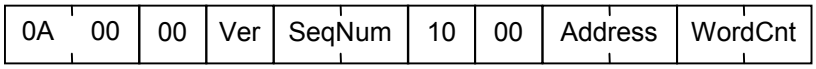
10 Read Device Registers (little Endian)

Description

This block reads from the device's registers. These registers are in a flat address space ranging from 0 to 65,535. On the RMC, these registers have pre-defined uses. Therefore, you must refer to the RMC Register Map in the RMCWin online help to read the data you wish.

Request

The request uses this format:

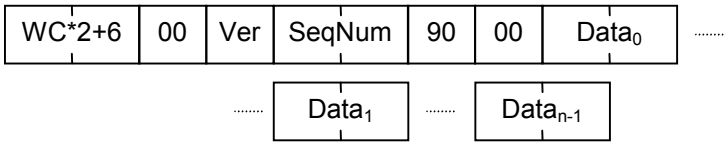


Address — The register address to begin reading from.

WordCnt — The number of *registers* to read (1-2048 in TCP, 1-512 in UDP).

Response

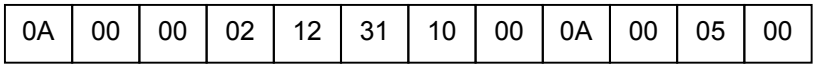
The success response follows this format:



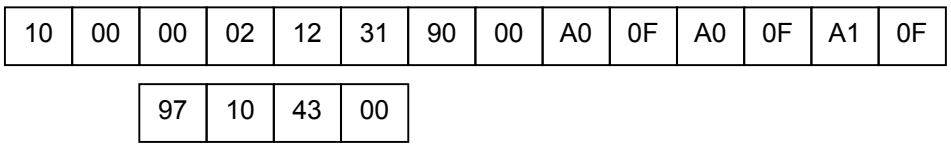
$Data_0$-$Data_{n-1}$ — The data read from the address requested.

Example

The Command Position, Target Position, Actual Position, Counts, and Status Word for axis 1 are located at register addresses 10-14. If a user wants to read these five registers, the following packet would be sent to the RMC:



If these fields have the values 4000 (0x0FA0), 4000 (0x0FA0), 4001 (0x0FA1), 4247 (0x1097), and 0x0043, the response would be the following:



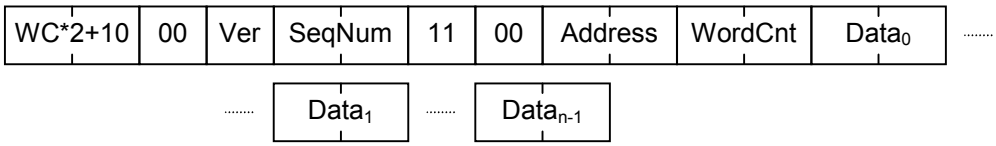
11 Write Device Registers (little Endian)

Description

This block writes to the device's registers. These registers are in a flat address space ranging from 0 to 65,535. On the RMC, these registers have pre-defined uses. Therefore, you must refer to the RMC Register Map in the RMCWin online help to write the data you wish.

Request

The request uses this format:



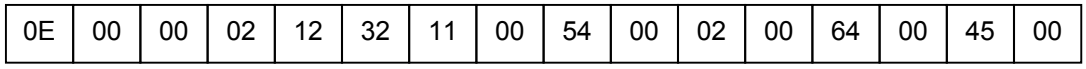
Address	The register address to begin writing to.
WordCnt	The number of *registers* to write (1-2048 in TCP, 1-512 in UDP).
$Data_0$-$Data_{n-1}$	The data to write to the address requested.

Response

The success response follows this format:

Example

To issue a Start Event (E) command to axis 0 with a command value of 100, the values 0x0064 and 0x0045 (ASCII for 'E') need to be written to registers 84 and 85 respectively. Therefore, the following packet would be sent to the RMC:



The response would be the following:

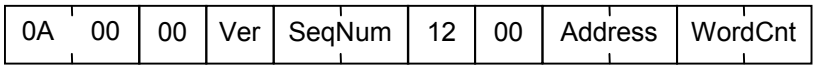
12 Read Device Registers (big Endian)

Description

This block is identical to block 0x10, except that the address, word count, and data are returned in big-Endian, or network, byte order. Notice that the Length field is still in little-Endian byte order.

Request

The request uses this format:

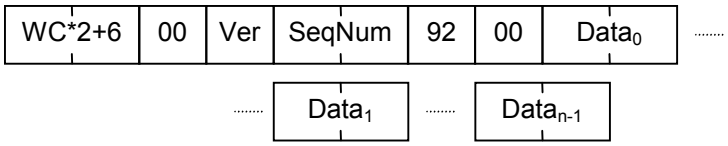


Address — The register address to begin reading from.

WordCnt — The number of *registers* to read (1-2048 in TCP, 1-512 in UDP).

Response

The success response follows this format:



$Data_0$-$Data_{n-1}$ — The data read from the address requested.

Example

The Command Position, Target Position, Actual Position, Counts, and Status Word for axis 1 are located at register addresses 10-14. If a user wants to read these five registers, the following packet would be sent to the RMC:

0A	00	00	02	12	33	12	00	00	0A	00	05

If these fields have the values 4000 (0x0FA0), 4000 (0x0FA0), 4001 (0x0FA1), 4247 (0x1097), and 0x0043, the response would be the following:

10	00	00	02	12	33	92	00	0F	A0	0F	A0	0F	A1

10	97	00	43

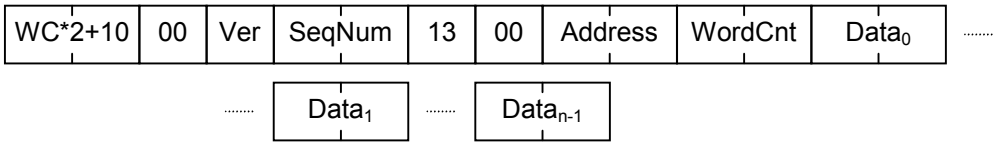
13 Write Device Registers (big Endian)

Description

This block is identical to block 0x11, except that the address, word count, and data are returned in big-Endian, or network, byte order. Notice that the Length field is still in little-Endian byte order.

Request

The request uses this format:



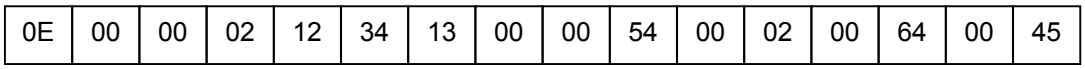
Address	The register address to begin writing to.
WordCnt	The number of *registers* to write (1-2048 in TCP, 1-512 in UDP).
$Data_0$-$Data_{n-1}$	The data to write to the address requested.

Response

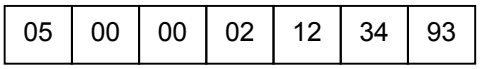
The success response follows this format:

Example

To issue a Start Event (E) command to axis 0 with a command value of 100, the values 0x0064 and 0x0045 (ASCII for 'E') need to be written to registers 84 and 85 respectively. Therefore, the following packet would be sent to the RMC:



The response would be the following:



Revision Log

The following changes have been made to this protocol:

Date	Author	Changes
July 1, 1999	QT	Original specification.
October 22, 1999	QT	Added functions 0x12 and 0x13 for network byte order. Increased the maximum length for TCP to 2048 words.
April 10, 2000	QT	Adopted the DMCP User Guide from the internal specification.
April 11, 2000	QT	Minor clarifications.
March 8, 2001	QT	Updated for version 2 of the packet format.
October 23, 2001	QT	Made minor corrections in cases where sometimes statements were made that were true for version 1 but not for version 2.