Use the Ethernet connections on Mint products to exchange data with a variety of third-party devices using the raw Ethernet terminal channel.

**Introduction**

The standard (non real-time) Ethernet ports on Mint controllers are often used in one of the following application cases:

- Connection of PROFINET (as part of a system comprising a third party PROFINET IO controller and MicroFlex e190 or MotiFlex e180 drives)
- Connection of Ethernet/IP (as part of a system comprising a third party Ethernet/IP master and MicroFlex e190 or MotiFlex e180 drives)
- Connection of Modbus TCP or Modbus UDP (as part of a system where NextMove e100, MicroFlex e190 or MotiFlex e180 are operating as Modbus servers, or where MicroFlex e190 or MotiFlex e180 are operating as Modbus clients)
- Connection of a host PC (where the PC may be issuing Mint commands directly to the NextMove e100, MicroFlex e190 or MotiFlex e180 using the Mint ActiveX controls via TCP/IP)

However, there is another way in which the standard Ethernet ports may be used to interface with third party products (e.g. PLCs, Vision systems) and that is by using these ports as a Mint 'terminal' channel (i.e. raw ASCII data can be received and transmitted via the standard Ethernet port).

The Mint Raw Ethernet Terminal uses port number 5002. This can be changed on NextMove e100 using the Mint keyword `TERMINALETHERTNETPORT…`

```plaintext
e.g. TERMINALETHERTNETPORT(_TERM4) = 502
```

For MicroFlex e190 and MotiFlex e180 drives the object dictionary (accessible via either the EtherCAT or Ethernet Powerlink page in Workbench) must be used to set this port number (object 0x4141:02).

**Note:** Do not change the port to 5000. This is reserved for the Mint ICM Ethernet port (the port used by Workbench and the Mint ActiveX control to send/receive Mint commands).

Possible uses for the Raw Ethernet terminal are:

- Implementation of protocol interpreters for standard Ethernet protocols (such as Modbus TCP, GE-SNPX etc.)
- Implementation of custom communication interfaces (e.g. a PLC with Ethernet communication abilities may be able to transmit/receive simple messages via Ethernet and a decoder written in Mint on the controller/drive can act on these). This can provide a very simple and cost effective means of interfacing a PLC with a Mint controller.
An input channel for data from a host program such as the HPGL interpreter (see [http://new.abb.com/motion/support/SupportMe/productsupport.asp?ID=HPGL](http://new.abb.com/motion/support/SupportMe/productsupport.asp?ID=HPGL)) or any other program that might typically send data via a serial buffer of some form

For this application note we will develop the idea of implementing a simple custom communication interface to illustrate the use of Raw Ethernet on Ethernet-equipped Mint products. The application note assumes the use of a MicroFlex e190 or MotiFlex e180 drive (with a fully commissioned motor) but the same principles apply to a NextMove e100 (although for a direct Ethernet connection to a NextMove e100 it must not be the manager of an Ethernet Powerlink network).

For convenience (e.g. to avoid the need for any PC-side programming) we will connect this to a PC and use Tera Term [https://ttssh2.osdn.jp/index.html.en](https://ttssh2.osdn.jp/index.html.en) as a means of sending and receiving the raw Ethernet data...

In practice a PC running under Windows would use the Mint ActiveX control to communicate with a Mint product but we'll use the PC and Tera Term to ‘simulate’ a non-Windows third party device such as a PLC or a Linux PC for example.

**Custom protocol definition**

**Data format**
For this application note we've invented a very simple ASCII style protocol as detailed below:

<table>
<thead>
<tr>
<th>Host/Client sends:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start Byte</strong></td>
</tr>
</tbody>
</table>

Controller replies to a valid write request with:

| **Start Byte** | **Function Bytes** | **Write Byte** | **Result Byte** *(ACK)* | **End Byte** |

Controller replies to a valid read request with:

| **Start Byte** | **Function Bytes** | **Read Byte** | **Result Byte** *(ACK)* | **Data Bytes** | **End Byte** |

Controller replies to an invalid read/write request with:

| **Start Byte** | **Function Bytes** | **Read/Write Byte** | **Result Byte** *(NAK)* | **End Byte** |

To keep the example as simple as possible we haven’t included a checksum in the message frame.

**Start byte**
The message frame starts with the ASCII character `[` – 5B hex. This is true for both the host/client message and the controller response.

The Mint program on the controller can therefore continually read data from the Raw Ethernet terminal using `x = INKEY(_TERM4)` until it detects the `[` character/byte.

If a start character from another supported protocol was used there is no need to disable the other protocol as the controller does not intercept any characters received via the Raw Ethernet terminal to check if they are part of any native protocols.
Function bytes
These bytes are used to tell the controller what type of function is to be performed (or to tell the host/client what function the response relates to). For this simple example we have defined just a small set of functions:

MVR – relative move (write only)
SPD – read/write speed
POS – read/write position

The data is case sensitive (e.g. the host must send “MVR” to issue a relative move and not “mvr”). The function bytes apply to both the host/client message and the Mint controller response.

Read/write byte
The host/client sets this byte to ‘=’ for a write or ‘?’ for a read

Data bytes
The host/client and Mint controller sends these bytes in a standard unpacked decimal fashion. So for example, “148” or “-233.56”. If the host/client is reading data then the data bytes field is not included in the message frame.

End byte
The message frame is terminated with the ] character/byte (5D hex). The Mint program is therefore able to sit in a loop collecting data until it detects the ] character, at which point it can then process the whole message to see what action is required.

Result byte
The Mint controller replies with [ACK] if a command from the host/client was successfully decoded and implemented or [NAK] if the command was invalid (e.g. unrecognized function type).

Example message transactions:

[MVR=12.3] Host tells controller to move relative 12.3 user units
[MVR=[ACK]] Controller accepts relative move command
[MVR=[NAK]] Reply if controller rejects move command
[SPD?] Host reads value of speed from controller
[SPD?[ACK]400] Controller returns value of 400 for current SPEED setting
[POS=77.2] Host tells controller to set Position of axis to 77.2
[POS=[ACK]] Controller accepts position write

Example Mint program
Included with this application note is a sample Mint program that interprets our custom protocol defined above. The core of this code is simply the receipt of characters via the raw Ethernet terminal…

nByteIn = INKEY(_TERM4)

…and the transmission of a complete string back to the host via the raw Ethernet terminal…

Print #_TERM4 sReply,

The comma on the end of the Print statement suppresses the carriage return that would normally be included automatically on the end of transmitted characters. This program can be run on NextMove e100, MicroFlex e190 or MotiFlex e180 (a Mint memory module must be installed on the drive products to allow Mint programming).
**Example Tera Term input**

With the Mint program running on your drive/controller and the drive enabled (use Workbench to enable the drive), start Tera Term and configure a connection as shown below…

![Tera Term New connection](image)

Ensure the Host setting matches the IP address of your connected drive/controller and that your PC’s network adaptor is on the same subnet (e.g. 192.168.0.x) as the drive/controller. Click OK and Tera Term should make a connection to the drive…

![192.168.0.1 - Tera Term VT](image)

You can now type in commands at the terminal (use Setup>Terminal… and select ‘Local echo’ if you want to see what you’re typing) and press Return and you will see the drive’s response. Here for example is what the drive replies with if we enter [SPD?]
You will notice that the Tera Term cursor remains at the end of the drive’s response (because we suppressed the carriage
return in the drive’s reply as described earlier). The protocol we’ve created doesn’t use the Carriage Return (or Line Feed)
characters so you could modify the Mint program’s PRINT statement in subroutine doReply to include the carriage return if you
think it helps to read the terminal…

‘Output message…
?#_TERM4 sReply

If you make this modification the terminal will now start new commands on a new line…

Contact us
For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC

© Copyright 2012 ABB. All rights reserved.
Specifications subject to change without notice.