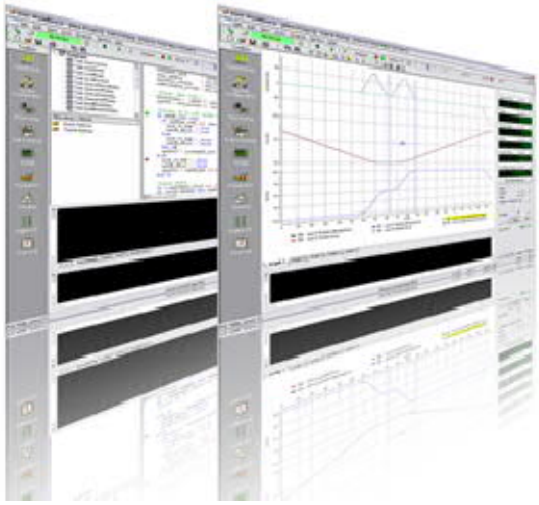
Motion Control Products

Application note
Host comms protocol 2

AN00129

Rev D (EN)

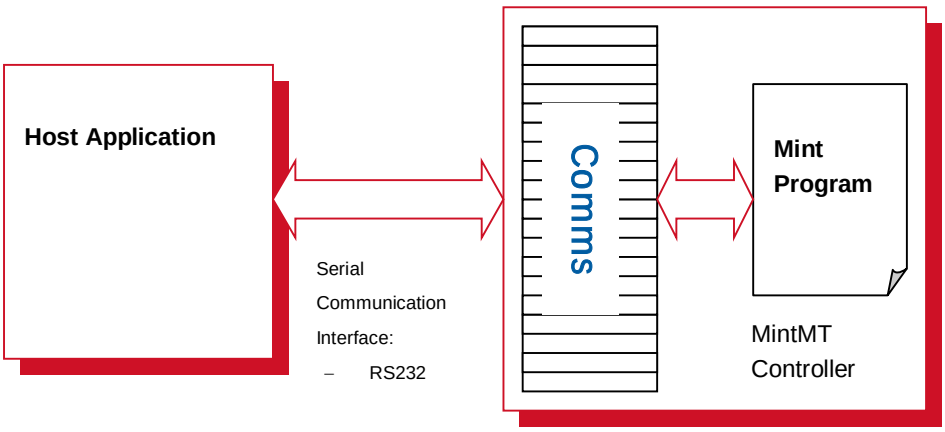
Host Comms Protocol (HCP) and Host Comms Protocol 2 (HCP2) provide an efficient way to transfer data bi-directionally between an MintMT controller's Comms array and a host application (where the 'host' is considered to be a PC, PLC or another controller capable of reading/writing ASCII data via a serial port). Refer to Application Note AN00110 for further details on Host Comms Protocol.



Introduction

The Comms array provides 99 data elements for use within any Mint program. These 99 elements can be used, for example, to transfer commands or recipes from the host or for the host to read status information from a MintMT controller (or multiple controllers if the host is connected to these via RS422). In addition to the 99 read/write elements, additional elements 100 - 255 are mapped onto system parameters such as position or I/O information. These additional elements form the 'Extended Comms array'.

In order to read from an element on a controller or write to an element, a serial data telegram is sent from the host. This is made up of ASCII characters, including control characters.



Whilst HCP only allows user access to the first 99 Comms elements, Host Comms Protocol 2 (HCP2) allows access to all 255 Comms elements. This makes it possible for a host system to read extended comms array data (position for example) from a controller that doesn't support MintMT programming (such as FlexDriveII).

A further restriction of HCP is that the host can only address nodes in the range 0 – 15. HCP2 raises this restriction to node addresses in the range 0 – 255.

HCP2 support has also been included on the MicroFlex range of AC Servo Drives, making it possible to either read drive status or send velocity or current demands via a serial connection. MicroFlex HCP2 operation is detailed in a later section of this application note.

Use of the Comms Array

As far as the MintMT program is concerned, Comms array elements look like just like any other array element. Comms differs from the normal MintMT array variables in that there is no need to define the 99 elements with a Dim statement since it is automatically defined.

Use of Comms is very simple. For instance, consider the following wire winding application, where a drum is traversed between two points defined by a host computer or PLC:

Example:

```
COMMS(3) = 0 'Initialize the value first
Loop
  If POS(0) > COMMS(2) Then Follow(0) = -COMMS(3)
  If POS(0) < COMMS(1) Then Follow(0) = COMMS(3)
  COMMS(4) = POS(0)
End Loop
```

The program is a simple continuous loop. The drum will be driven back and forth between points specified by Comms(2) and Comms(1) at a speed proportional to the rotation of the drum (measured using an encoder on the drum). The following ratio is specified by the variable Comms(3). These values can be changed at any time by sending data packets from the host and are automatically made available to the Mint program. At the same time the host is able to read back the axis position by reading the value stored in Comms(4).

Example:

```
Dim nCommand As Integer

COMMS(1) = 0
Loop
   nCommand = COMMS(1)
   If nCommand <> 0 Then
      IF nCommand = 1 THEN SPEED(1) = COMMS(2) : MOVEA(1) = COMMS(3) : GO(1) : PAUSE IDLE(1)
      IF nCommand = 2 THEN SPEED(1) = COMMS(2) : MOVER(1) = COMMS(3) : GO(1) : PAUSE IDLE(1)
      IF nCommand = 3 THEN HOME(1)  = COMMS(4) : PAUSE IDLE(1)
   End If
   COMMS(1) = 0 : 'Host can see when command is complete when Comms(1) is zero
End Loop
```

In this example, the host will transmit a command to Comms(1). The data for the move type is held in Comms elements 2 to 4. When the move is complete, the command is set to zero. This acts as an acknowledgement to the host.

Events

When any of Comms elements 1 through to 5 are written to by the host, this will result in a MintMT event being executed if it exists (even if the data value itself has not changed).

```
Event  comms1
    JOG(0)  =  COMMS(1)
End  Event
```

In this example, the motor will jog at a speed defined by Comms(1) and the jog speed will be modified every time the host sends a value to Comms(1).

Extended Comms Array

In addition to the 99 read/write elements of the comms array, additional elements 100 - 255 are mapped onto system parameters such as position or I/O information. These additional elements form the extended comms array and can be accessed via HCP2.

The mapping allocates space for up to 15 axis parameters for a maximum of eight axes, and an additional 36 elements for miscellaneous I/O data. The mapping of these parameters is indicated below. The first table shows the elements for the axis parameters, with the second table showing the offset for the system parameter (Note that MicroFlex adopts a slightly modified version of these mappings – this is described later):

Comms elements	Contents
100 .. 114	Axis 0
115 .. 129	Axis 1
130 .. 144	Axis 2
145 .. 159	Axis 3
160 .. 174	Axis 4
175 .. 189	Axis 5
190 .. 204	Axis 6
205 .. 219	Axis 7
220 .. 255	I/O and miscellaneous data

The following table indicates the offset of data mapped for each axis:

Comms element offset	Data	Read / Write	Equivalent MintMT Keyword
0	Actual position	R	POS
1	Actual velocity	R	VEL
2	Mode of motion	R	AXISMODE
3	Axis Error	R	AXISERROR
4	Drive demand	R	DAC * See restrictions
5	Encoder position	R	ENCODER
6	Motor in position	R	IDLE
7	Positional Error	R	FOLERROR
8 .. 14	Reserved	--	

Example: A host could read the encoder value for Axis 0 by sending a HCP2 data telegram to read the contents of comms array element 105.

The following table indicates the offset for the I/O and miscellaneous data:

Comms element offset	Data	Read / Write	Equivalent MintMT Keyword
0	Digital inputs	R	IN
1	Reserved	--	
2	Digital outputs	R/W	OUT
3	Reserved	--	
4	Analog channel 0	R	ADC(0)
5	Analog channel 1	R	ADC(1)
6	Analog channel 2	R	ADC(2)
7	Analog channel 3	R	ADC(3)
8	Analog channel 4	R	ADC(4)
9	Analog channel 5	R	ADC(5)
10	Analog channel 6	R	ADC(6)
11	Analog channel 7	R	ADC(7)
12	MintMT error	R	Err
13	MintMT error line number	R	Erl
14	MintMT axis error	R	ErrAxis
15	MintMT execution status. 0: No program running 1: Program running	R	
16	Reserved	R	
17	MintMT line number	R	
18	Reserved		
19	Firmware Build Number	R	FIRMWARERELEASE
20 .. 35	Reserved	--	

Refer to the MintMT Help file for further details on the Extended Comms array including information on controller specific limitations.

Example: A host could read the state of the outputs on bank 0 by sending a HCP2 data telegram to read the contents of comms array element 222.

Writing Data

The host sends the following HCP2 data telegram to write information to the card (control characters are shown in shaded boxes).

Reset	BEL (07 Hex)
Controller Node ID MSD	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the Most Significant digit of the controller's Node ID. Lower case a to f is invalid and will result in the controller ignoring the message (as the encoded ID can never match the controller's Node ID)
Controller Node ID LSD	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the Least Significant digit of the controller's Node ID. Lower case a to f is invalid and will result in the controller ignoring the message (as the encoded ID can never match the controller's Node ID)
Start	STX (02 Hex)
Comms Element MSD	ASCII character 0-9; A-F (30 to 39 Hex; 41 to 46 Hex); Hex representation of the Most Significant Digit of the Comms array element
Comms Element LSD	ASCII character 0-9; A-F (30 to 39 Hex; 41 to 46 Hex); Hex representation of the Least Significant Digit of the Comms array element
Data Field	Up to 127 ASCII characters (65 characters for Series 2 drives), decimal format, can be optionally comma separated in order to write to more than one location in a single transaction (e.g. "1234,-45.6,11.2"). Each data value is loaded into subsequent comms elements (e.g. if the data string "10,20,30" is sent to Comms element 10, element 10 will contain 10, element 11 will contain 20 and element 12 will contain 30). The last character in the string must be numeric. NAK will be returned if message attempts to access comms elements higher than 255 or read only elements. Leading spaces are thrown away. Spaces in the data will terminate the conversion. Data is accurate to 4 decimal places with rounding down/up on the 5th decimal place. Leading zeroes can be omitted (e.g. .5 would be a valid value). Positive values can be optionally preceded with the + character.
End	ETX (03 Hex)
Checksum	1 byte, XOR of everything after (but not including) BEL, up to and including ETX

Note that the Controller Node ID and the comms element are both hex-encoded ASCII values. To write to comms element 38 for example, the user would send the ASCII character 2, followed by the ASCII character 6 ($38_{dec} = 26_{hex}$).

Providing the Node ID of the controller receiving the data packet matches the Controller Node ID transmitted as part of the message, the controller will respond with an ACK (06 Hex) or a NAK (15 Hex) depending on the validity of the rest of the message. The reply will normally be returned within 50ms of receipt of the checksum. However, hardware handshaking and other processes (such as CAN operations) could delay the reply by 100mS.

A NAK will be returned if :

- the data field string is not formatted as described by the table above
- the message attempts to write to a comms element greater than 255 (e.g. starts to write data to comms element 254 and the data field contains 3 values)
- the message attempts to write to a comms element less than 1
- the message attempts to write to a read only comms element
- the checksum is incorrect.

Note that in instances where data has to be processed before a possible error with the telegram is detected the controller would write valid data to it's comms array before it returned a NAK (i.e. the ACK/NAK is an indication of the overall success of the transaction). For example, writing 2 values starting at comms element 99 would result in an attempt to write to comms element 100, which is read only. The first data field element would be written to comms element 99 and then the controller would return NAK on attempting to write the second data field element to comms element 100.

HCP2 writes initiated via the ActiveX control (using a new function – SetCommsWriteHCP2) do not retry if a NAK is received (this is down to the user to code this functionality). The ActiveX interface returns an error code to indicate reception of a NAK. This is described in a later section of this document.

If the telegram is formatted correctly, and can be decoded by the controller, an ACK will be returned.

Reading Data

The host sends the following HCP2 data telegram to read data (control characters are shown in shaded boxes):

Reset	BEL (07 Hex)
Controller Node ID	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the Most Significant digit of the controller's Node ID. Lower case a to f is invalid and will result in the controller ignoring the message (as the encoded ID can never match the controller's Node ID)
Controller Node ID	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the Least Significant digit of the controller's Node ID. Lower case a to f is invalid and will result in the controller ignoring the message (as the encoded ID can never match the controller's Node ID)
Start	STX (02 Hex)
Comms Element MSD	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the Most Significant digit of the controller's Comms array element. Lower case a to f is invalid and will result in the controller returning NAK
Comms Element LSD	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the Least Significant digit of the controller's Comms array element. Lower case a to f is invalid and will result in the controller returning NAK
End	ENQ (05 Hex)
Checksum	1 byte, XOR of everything after (but not including) BEL, up to and including ENQ

The controller will respond with the following data packet if the received data telegram is valid:

Start	STX (02 Hex)
Comms Element MSD	ASCII character 0-9; A-F (30 to 39 Hex; 41 to 46 Hex); Decimal representation of the Most Significant Digit of the Comms array element
Comms Element LSD	ASCII character 0-9; A-F (30 to 39 Hex; 41 to 46 Hex); Decimal representation of the Least Significant Digit of the Comms array element
Data Field	Up to 20 characters in decimal format. Data is accurate to 4 decimal places with rounding down/up on the 5th decimal place.
End	ETX (03 Hex)
Checksum	1 byte, XOR of everything after (but not including) STX, up to and including ETX

The reply will normally be returned within 50ms of receipt of the checksum character. However, hardware handshaking and other processes (such as CAN operations) could delay the reply.

In place of the telegram detailed above, only NAK will be returned if :

- the Comms element is invalid (not in the range 01 to 255 or contains lower case characters)
- a read of a write only element is attempted
- the checksum is invalid

HCP2 reads initiated via the ActiveX control (using a new function - GetCommsReadHCP2) do not retry if a NAK is received (this is down to the user to code this functionality). The ActiveX interface does not return an error code to indicate reception of a NAK. Errors are limited to Timeout (which is what will happen if NAK is returned) and Invalid Data (if more than 20 characters are received as part of the data field). Operation of HCP2 via the ActiveX control is described later in this document.

HCP2 between Controllers

The MintMT COMMSMODE keyword allows the user to determine whether HCP or HCP2 data telegrams are utilized when MintMT controllers are reading/writing comms array elements on other MintMT controllers on a RS485 network. Using HCP2 allows a MintMT controller to access the Extended Comms array data on other controllers as described previously.
Bit 2 of the COMMSMODE keyword specifies the setting:

0 – HCP operates (Nodes 0 – 15, Comms elements 01-99)
1 – HCP2 operates (Nodes 0 – 255, Comms elements 01-255)

PC Host and ActiveX Control

If the host is a PC then, given the simplicity of Host Comms Protocol 2, it can be easily implemented under any operating system that provides access to the PC's available serial ports (e.g. Microsoft Windows ®, Linux, QNiX etc..).

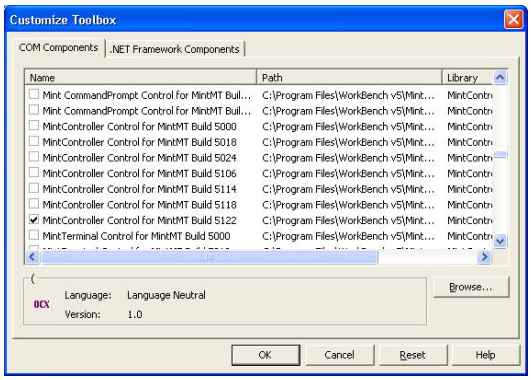
However, to aid development under the Microsoft Windows ® environment, ABB provides an ActiveX control, installed with Workbench v5, that encapsulates the Host Comms Protocol 2 functionality into a simple to use ActiveX method.

Example

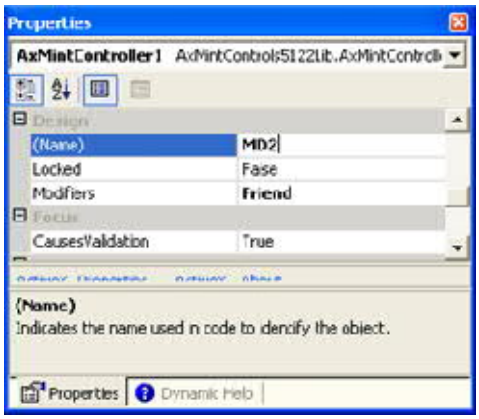
Our application requires that a PC host application (coded in Visual Basic.net ®) running under Microsoft Windows XP needs to read and write data from/to the Comms array on a MintDrive*II* via an RS232 serial port.
Firstly include the ActiveX control in a new VB project:

- Right-click the Toolbox and select Customise Toolbox…

- From the list of available components select the latest version of the MintController Control for MintMT

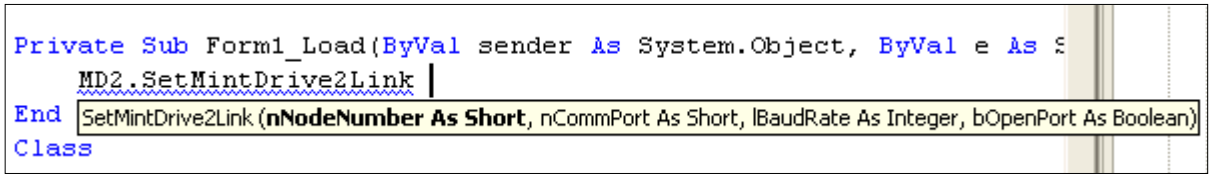


- An icon for this control will appear at the bottom of the list of components in the Toolbox. Scroll down to this and double-click it to add it to the available form
- VB.net will name this control axMintController1 by default – we will rename this to MD2 via the Properties dialogue provided by VB.net



Now we need to establish a connection (open the serial port) to the MintDriveII:

- Double-click the VB form (the code window for the Form Load event will open)
- In this event start typing md2. (as soon as the period is entered VB will automatically display a list of available ActiveX methods and properties) followed by setmintdr (you will see that as you keep typing VB scrolls through the list automatically to find a matching method or property)
- As soon as VB has highlighted setMintDrive2Link press the space bar to auto-complete the command – VB will now provide a tooltip listing the parameters that need to be entered with this method

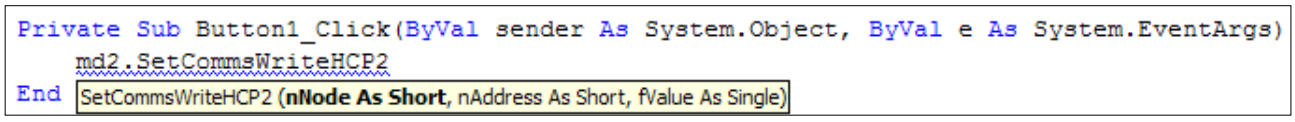


- Our drive is configured as Node 2, our PC is using Com1, the drive is configured to operate at 57600 baud and we'd like to open the port now so we complete the method by typing:

  2,1,57600,True

This is all the code we need to get VB to communicate with the drive. Now we'll add a Command button to read and write some data from/to the drive using Host Comms Protocol 2:

- Double-click the Button control on the toolbox and position the button on the form
- Use the Properties dialogue to change the button text to Read/Write
- Double-click the button to call up the Click procedure for the button
- Start typing md2. (again VB will automatically display a list of available ActiveX methods and properties). This time the method we want to use is setCommsWriteHCP2 so start typing this in and press the space bar once the method is



highlighted.

- We now need to enter the Node address, the Comms element (1 to 99) that we want to write data to and enter a value to write (we'll enter 2, 1, 123.456)

This command could be considered to be complete (when we click on the button at run-time the ActiveX control will format the HCP2 message for us and send the resulting ASCII data telegram to the drive), however ideally we need to check the return from this method to see whether data was written successfully or not (i.e. whether the drive returned ACK or NAK).
In VB.net we achieve this by adding an error handler to our procedure (this is illustrated by the code snippet below) :

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        On Error GoTo ErrHandler
        md2.SetCommsWriteHCP2(2, 1, 123.456)
        Exit Sub
ErrHandler:
        MsgBox("Error communicating with drive : " & Err.Description)
    End Sub
```

It is common practice to retry a number of times before reporting an error – this is easily incorporated into our error handling routine if required.
Now we'll add the ActiveX method to read data from the drive via Host Comms Protocol 2. We'll read Comms(1) – this way we can read back what was written !

- Declare a single precision floating point value called pfValue (Dim pfValue As Single)
- Add the ActiveX method MD2.GetCommsReadHCP2 (2, 1, pfValue) just after the SetCommsWriteHCP2 call
- Switch to form design and double-click the label control on the VB.net toolbox to add a label to the form – move this to an appropriate position on the form
- Use the Properties dialogue to rename this label lblComms1 and change the initial text to 0.0
- Switch back to the code window for the button click event and add the following code to update the label text:

  lblComms1.Text = pfValue

The resulting code should appear as follows:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs
        On Error GoTo ErrHandler
        Dim pfValue As Single
        md2.SetCommsWriteHCP2(2, 1, 123.456)
        md2.GetCommsReadHCP2(2, 1, pfValue)
        lblComms1.Text = pfValue
        Exit Sub
ErrHandler:
        MsgBox("Error communicating with drive : " & Err.Description)
    End Sub
```

Compile and run this application and click the button on the form. The application should write 123.456 to the MintDrive[II] and the label text should update to confirm this.

Disconnect the serial lead between the PC and the drive to confirm operation of the error handler. A dialogue similar to the one shown below should appear after a short delay:

As the ActiveX server (MilServer5000) is transaction based it is also possible to run Workbench v5 simultaneously to verify operation of the SetCommsWrite instruction (via the Comms Watch Window on the Spy pane).

Comms Array Access via ABB Binary Protocol 2 (BBP2)

In addition to the ActiveX methods described above that implement Host Comms Protocol 2, there are two further ActiveX methods that provide an alternative, more robust, way of reading and writing data from/to the controllers Comms array. These methods use BBP2 (as does Workbench v5 itself) to construct the binary data telegrams.

The code format is very similar to that illustrated previously. The examples below illustrate how our MD2 object may read and write data using the BBP2 Comms methods:

Writing Data:
MD2.Comms(1) = 123.456

Reading Data:
Dim pfValue As Single
pfValue = MD2.Comms(1)

Use of HCP2 and Extended Comms Array on MicroFlex Drive

In some applications it may be useful to be able to communicate with a MicroFlex drive via the serial comms port in order to:

- Read status information such as trip status, motor currents, speeds etc.
- Send torque or speed references.

Whilst this is possible via the ActiveX interface, this requires a PC running a Microsoft operating system and ABB software. Some customers may prefer to write their own communications software or use a non-Windows PC or other device such as a PLC to communicate with the drive.

HCP2 allows read/write access to all 255 Comms array elements. Normally a Mint program processes data in the first 99 of these elements. For example, if the customer requires drive current information via Comms, a Mint program would be written to regularly update a given Comms element with the drive current value.

Since MicroFlex does not support Mint programs, a different system is required. As a number of the Extended Comms array elements are already predefined to refer to specific quantities (for example, on Series II drives and NextMove motion controllers, reading Comms element 100 will give the current position of axis 0) it is sensible that MicroFlex makes use of this Extended

Comms array to map drive parameters such as drive current and speed reference to some of the Extended Comms array elements unused by other controllers. This functionality is available to MicroFlex users in firmware builds 5221 onwards.

A host system can therefore read/write these drive parameters by sending the appropriate HCP2 data telegram.

MicroFlex Extended Comms Array Mappings

The table below details the mapping of axis related MicroFlex drive parameters (only elements that are actually used are shown):

Comms element	Mapping	Read/Write	Description/Units
100	Position	R	Position in counts (not possible to apply SCALEFACTOR on MicroFlex)
103	Axis Error	R	Bitmap containing current axis errors
107	Following Error	R	Following error in counts (not possible to apply SCALEFACTOR on MicroFlex)
108	Command Word	R/W	A bitmap containing drive enable, stop and cancel bits
109	Status Word	R	A bitmap containing status information such as drive enable state
110	Drive Error	R	Bitmap containing current drive errors
111	Demand Position	R	Position demand in counts (not possible to apply SCALEFACTOR on MicroFlex)
116	Torque Ref	R/W	Torque reference in % of drive rated current
117	Torque Demand	R	Torque demand in % of drive rated current
118	Speed Ref	R/W	Speed reference in % of application max speed
119	Speed Demand	R	Speed demand in % of application max speed
120	Speed Measured	R	Measured speed in % of application max speed
121	Speed Error	R	Difference between demand and measured speed in % of application max speed
124	Drive Overload Area	R	Percentage into a drive overload condition
125	Motor Overload Area	R	Percentage into a motor overload condition
126	Bus Voltage	R	DC bus voltage in volts
127	Current Measured	R	RMS drive current in Amps
128	U Phase Current	R	U phase current in Amps
129	V Phase Current	R	V phase current in Amps
130	Hall State	R	Hall state bitmap
131	Absolute Encoder	R	Absolute position in counts (for SSI encoders).
132	Phase search status	R	Current status of phase search

Notes:
Attempting to write to one of the read-only elements will cause HCP2 to return a NAK.

A value written to comms element 116 (Torque Reference) will only be applied if the drive is in current control mode and the reference source (TORQUEREFSOURCE) is set to “host”. Otherwise HCP2 will return a NAK.

A value written to comms element 118 (Speed Reference) will only be applied if the drive is in speed control mode and the reference source (SPEEDREFSOURCE) is set to “host”. Otherwise HCP2 will return a NAK.

The table below details the mapping of miscellaneous and I/O related MicroFlex drive parameters (only elements that are actually used are shown):

Comms element	Mapping	Read/Write	Description
220	Digital Inputs (Bank 0)	R	General purpose digital input status
222	Digital Outputs (Bank 0)	R	Digital output status (i.e. global error output)
224	Analog Input (Channel 0)	R	Analog input value in %
239	Firmware Build Number	R	Firmware build number
242	Temperature	R	Drive temperature in degrees C

Note: Attempting to write to one of these read-only elements will cause HCP2 to return a NAK.

MicroFlex Command Word

The command word (element 108) is a bitmap that is used to control the fundamental operation of the drive. Individual bits are assigned to match the format of the first byte of the command telegram used by DeviceNet (refer to MintMT Help file for further details).
The bits defined for MicroFlex are:

Bit	Purpose	Description
0-5	Undefined	These are used either for DeviceNet specific functions or for controlling functionality not present on MicroFlex.
6	Cancel	Setting this bit to 1 will attempt to cancel any asynchronous axis errors. Setting bit to 0 will allow motion to proceed again. The cancel bit should not be left high as this will interfere with the normal process of latching errors.
7	Drive enable	Setting this bit to 1 will enable the drive if there are no axis errors and the hardware enable is active. Setting this bit to 0 will disable the drive.

Reading from this comms element will give the current command word settings.

MicroFlex Status Word

The status word (element 109) is a bitmap that is used to determine the operating status of the drive. Individual bits are assigned to match the format of the first 2 bytes of the response telegram used by DeviceNet (refer to MintMT Help file for further details).

The bits defined for MicroFlex are:

Bit	Purpose	Description
0-2	Undefined	DeviceNet specific or unsupported functions.
3	Asynchronous error	This bit will be set if the drive has an asynchronous or drive error.
4	Undefined	Unsupported function.
5	Error or stop input status	This bit will be set if the general purpose input is set up to be an error or stop input and the input is active.
6	Hardware drive enable status	This bit will be set if the hardware drive enable input is active.
7	Drive enabled	This bit will be set if the drive is enabled (i.e. both the hardware and software enables are high and there are no errors).
8-15	Undefined	Unsupported functions.

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