

# Mint<sup>Multi-Tasking</sup> MT Application Note

## AN00110-003 – Serial Host Comms Protocol

### Related Applications or Terminology

- **Computer to Controller communications**
- **Host to Controller communications**
- **Data transfer from host computer**
- **COMMS() array access**

### Overview

The Host Comms Protocol (HCP) provides an efficient way to transfer data bi-directionally between an executing Mint program'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).

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).

With MintMT events, it is also possible for a MintMT application to be interrupted when new data is posted, by an external source, into Comms elements 1 through to 5.

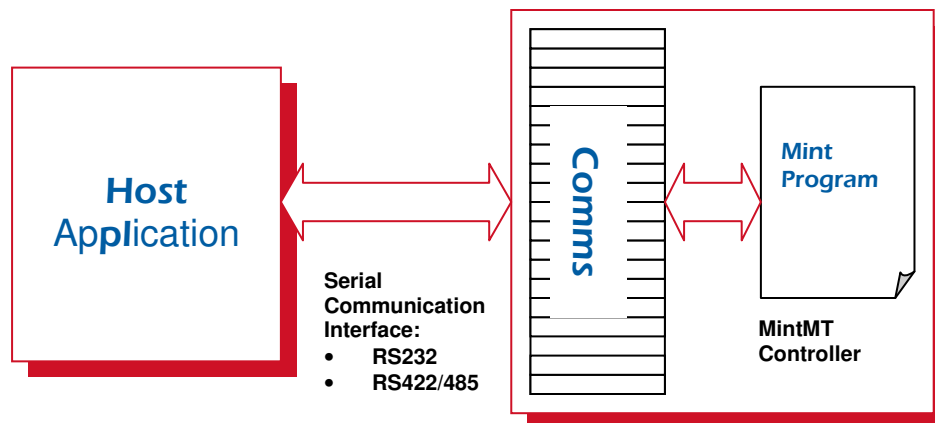
In order to read from an element on a controller or write to an element, a serial data telegram is sent from the host. This is made up of ASCII characters, including control characters. The MintMT Host Communications Protocol is based on ANSI x3.28 allowing it to be easily integrated into host computers or PLCs.

### Supported Controllers

<b>NextMovePCI</b>	<input checked="" type="checkbox"/>
<b>NextMoveBX<sup>II</sup></b>	<input checked="" type="checkbox"/>
<b>NextMoveST</b>	<input checked="" type="checkbox"/>
<b>NextMoveES</b>	<input checked="" type="checkbox"/>
<b>NextMoveESB</b>	<input checked="" type="checkbox"/>
<b>NextMove100</b>	<input checked="" type="checkbox"/>
<b>MintDrive<sup>II</sup></b>	<input checked="" type="checkbox"/>
<b>Flex+Drive<sup>II</sup></b>	<input checked="" type="checkbox"/>

### Relevant Keywords

**Comms() array**  
**Event commsX**



# Mint<sup>MT</sup> Multi-Tasking 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
```

# Mint<sup>MT</sup> Multi-Tasking Application Note

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).

## Writing Data

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

Reset	EOT (04 Hex)
Controller Node ID	ASCII character 0-9 (30 to 39 Hex);A-F (41 to 46 Hex); Hex representation of the controller's Node ID. Lower case a to f is invalid – e.g. the host must address Node 13 as Node D (44 Hex). Node 13 will ignore any telegram encoded with a Node ID of d (64 Hex).
Controller Node ID	Node ID character repeated
Start	STX (02 Hex)
Comms Element MSD	ASCII character 0-9 (30 to 39 Hex); Decimal representation of the Most Significant Digit of the Comms array element
Comms Element LSD	ASCII character 0-9 (30 to 39 Hex); Decimal representation of the Least Significant Digit of the Comms array element
Data Field	Up to 60 ASCII characters, decimal format, can be optionally comma separated in order to write to more than one element in a single transaction (e.g. "1234,-45.6,11.2"). Each data value is loaded into subsequent comms elements. The last character in the string must be numeric. NAK will be returned if telegram attempts to access comms elements higher than 99 or there are more than 60 characters in the data field. 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) STX, up to and including ETX

Note that the Controller Node ID and the comms element are both ASCII values. To write to comms element 12, the user would send the ASCII character 1, followed by the ASCII character 2.

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 contains an invalid format string, the telegram attempts to write to a comms element greater than 99 or less than 01, or the checksum is incorrect.

**Mint<sup>MT</sup> Multi-Tasking** Application Note

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 data telegram to read data (control characters are shown in shaded boxes):

Reset	EOT (04 Hex)
Controller Node ID	ASCII character 0-9 (30 to 39 Hex); A-F (41 to 46 Hex); Hex representation of the controller's Node ID. Lower case a to f is invalid – e.g. the host must address Node 13 as Node D (44 Hex). Node 13 will ignore any telegram encoded with a Node ID of d (64 Hex).
Controller Node ID	Node ID character repeated
Start	STX (02 Hex)
Comms Element MSD	ASCII character 0-9 (30 to 39 Hex); Decimal representation of the Most Significant Digit of the Comms array element
Comms Element LSD	ASCII character 0-9 (30 to 39 Hex); Decimal representation of the Least Significant Digit of the Comms array element
End	ENQ (05 Hex)

Providing the Node ID of the controller receiving the data packet matches the Controller Node ID transmitted as part of the message and the Comms element to be read is in the range 01 to 99 then the controller will respond with the following data telegram:

Start	STX (02 Hex)
Comms Element MSD	ASCII character 0-9 (30 to 39 Hex); Decimal representation of the Most Significant Digit of the Comms array element
Comms Element LSD	ASCII character 0-9 (30 to 39 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 ENQ character. However, hardware handshaking and other processes (such as CAN operations) could delay the reply by 100ms.

In place of the telegram detailed above, only NAK will be returned if the Comms element is invalid (not in the range 01 to 99).

# Mint<sup>MT</sup> Multi-Tasking Application Note

## PC Host and ActiveX Control

If the host is a PC then, given the simplicity of the serial Host Comms Protocol, 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..).

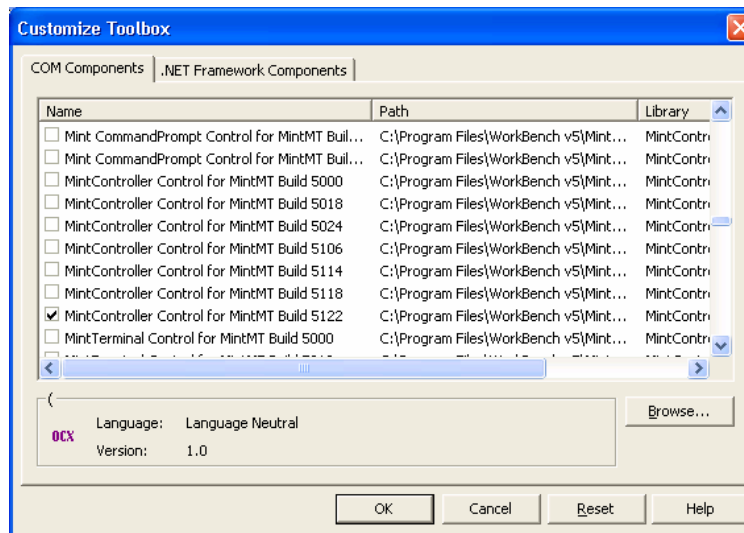
However, to aid development under the Microsoft Windows ® environment, Baldor provides an ActiveX control, installed with Workbench v5, that encapsulates the Host Comms Protocol functionality into a simple to use ActiveX method.

## Example

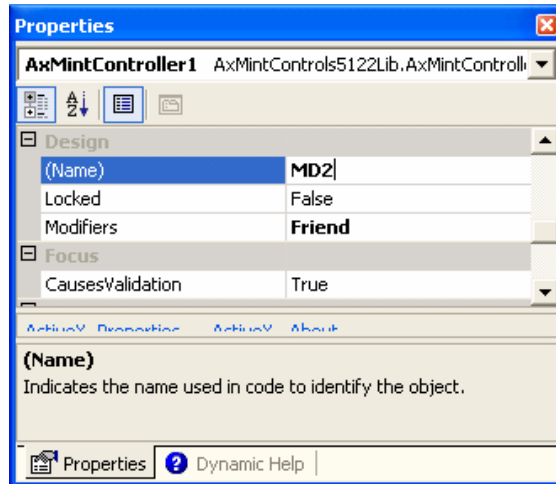
Our application requires that a PC host application (coded in Visual Basic.net ® ) running under Microsoft Windows XP needs to read and write data from/to the Comms array on a MintDrive<sup>II</sup> via an RS232 serial port.

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

**Mint<sup>MT</sup>** Multi-Tasking  
**Application Note**

Now we need to establish a connection (open the serial port) to the MintDrive<sup>II</sup>:

- 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

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As EventArgs)
    MD2.SetMintDrive2Link
End Sub
Class
```

- 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:

- 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

**Mint<sup>Multi-Tasking</sup> MT Application Note**

- 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 setCommsWrite so start typing this in and press the space bar once the method is highlighted.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    MD2.SetCommsWrite
End Sub
    SetCommsWrite (nAddress As Short, fValue As Single)
..
```

- We now need to enter the Comms element (1 to 99) that we want to write data to and enter a value to write (we'll enter 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 HCP 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 ErrorHandler
    MD2.SetCommsWrite(1, 123.456)
    Exit Sub

ErrorHandler:
    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. 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.GetCommsRead (1, pfValue) just after the SetCommsWrite 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
```

# Mint<sup>MT</sup> Multi-Tasking Application Note

The resulting code should appear as follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim pfValue As Single
    On Error GoTo ErrorHandler
    MD2.SetCommsWrite(1, 123.456)
    MD2.GetCommsRead(1, pfValue)
    lblComms1.Text = pfValue
Exit Sub
```

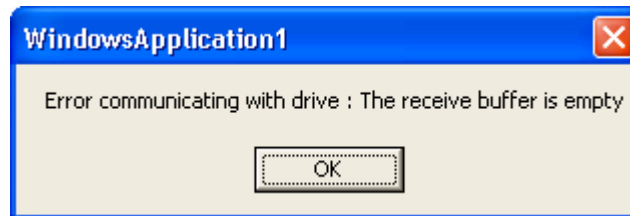
ErrorHandler:

```
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<sup>II</sup> 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 Baldor Binary Protocol 2 (BBP2)

In addition to the ActiveX methods described above that implement the serial Host Comms Protocol, 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)
```