

Mint^{MT} Multi-Tasking
Application Note**AN00134-001: Handling Recipe Downloads from CANopen HMI****Overview**

Baldor's Human-Machine Interface (HMI) panels are available as serial RS232/485 devices as standard (refer to Application Note AN00112 for further details about the operation of the serial HMI).

However, the panels can be modified to operate on a CANopen network by fitting option card KPD-OPTC.

In Application Note AN00111 we looked at the essential operating elements of the CANopen HMI and how this interfaced with a Mint controller. In this Application Note we will look further at how the HMI notifies the Mint controller that new data has been entered and how the Mint program should be structured to cope with large quantities of data (i.e. Recipe data) being made available at the same time.

Application Note AN00137 also provides useful information on the use of Recipes with Baldor HMIs.

Read/Write Data

Firstly let's recap on adding read/write data to the HMI project (i.e. data values that can either be updated by the Mint controller or that can be entered by the user).

Ensure Workbench v5 is closed and start the Baldor HMI software (refer to Application Note AN00111 for further information on initially configuring an HMI project to suit the panel model being used).

The procedure for adding Read/Write data to the HMI is very simple. First we need to add a label to the screen for the read/write data – we'll display a 'Product Length' setting that can be edited on the panel. Click where the label should start and type Length. The text will appear on the HMI screen design area.

Now click on the Numeric Field button in the toolbox and click and drag where the numeric data is to be displayed to display the Numeric Field Properties dialog.

This field will be a floating-point value (using 1 decimal place). We will make it the first element of the floating point database so the Mint controller will address it as Comms(2,255). Selecting Read/Write access will allow us the option to enter minimum and maximum data entry values – we'll limit this to the range 1.0 to 1200.0.

Supported Controllers

NextMove PCI	<input checked="" type="checkbox"/>
NextMove BX ^{II}	<input checked="" type="checkbox"/>
NextMove ST	<input checked="" type="checkbox"/>
NextMove ES	<input checked="" type="checkbox"/>
NextMove ESB	<input checked="" type="checkbox"/>
NextMove E100	<input checked="" type="checkbox"/>
MintDrive ^{II}	<input checked="" type="checkbox"/>
Flex+Drive ^{II}	<input checked="" type="checkbox"/>

Note – **Flex+Drive^{II}** is a Slave only. There must be another master node in the Network e.g. **MintDrive^{II}**

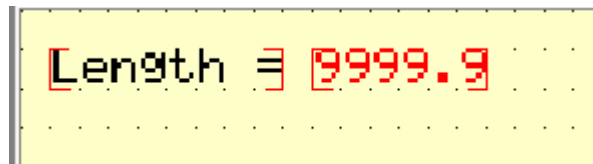
Relevant Keywords**COMMS****CONNECT****BUSEVENT****BUSEVENTINFO**

Mint^{MT} Multi-Tasking Application Note

The table below details the main settings for the dialog:

Parameter	Value
Style	Numeric
Format	Decimal Unsigned, Fixed Point 1
Size	6
Reference	PLC
Data type	Float_db
Index	1 i.e. Comms(255)
Data Format	Float
Access	Read/Write
Scaling	A1=1, A2=1, B=0
Min Value	Constant : 1.0
Max Value	Constant : 1200.0

Click on OK and the HMI software will now display 9999.9 to indicate where our numeric data will appear in our application.



Place the panel into configuration mode (hold the bottom right-hand corner of the touch screen or press the enter key for three seconds) and then download the HMI application.

Important - Save the HMI project.

Assuming that the panel has been configured as CANopen Node 2, we can now test the operation of our HMI project by reading the value of Comms(255) stored on the HMI by querying the value of Comms(2,255) from the Mint controller (e.g. via the command prompt in Workbench v5). Because downloading the HMI project interrupted the operation of the panel we need to reinitialize the panel and re-establish the CANopen connection first...

```
BUSBAUD (1) = 500      `Set baud rate to 500kbaud
BUSRESET (1)          `Reset the Canbus controller
NODETYPE (1,2) = 0    `Remove node 2 from bus
NODESCAN (1,2)       `Detect node 2
PAUSE NODELIVE (1,2) `Wait for node to be detected
CONNECT (1,1,2) = 1  `Establish a connection
```

Now enter some data on the panel for Product Length (press the number on the touch screen or press the data entry key on the panel keypad). Having entered some data enter the following text at the Mint command prompt...

```
Print COMMS (2,255)
```

The panel should respond with the data previously entered for Product Length.

Mint^{Multi-Tasking} MT Application Note

As we saw in Application Note AN00111, our Mint program can either poll this location (in a Loop...End Loop construct for example) and take action as required if the value changes or (more elegantly) the BUS1 event can be used to make detection of a change in HMI value an event driven process.

Event Driven Operation

The HMI automatically generates a bus event of type 12 (pre-defined Mint constant `_bethMI_COMMS_UPDATE`) whenever the user modifies any of the numeric data stored on the panel. As we are using CANopen (Bus 1) this can be detected by adding a BUS1 event to a Mint program and querying the value of `BUSEVENT(1)`

In addition, the HMI panel also provides details of which data element (comms element) was changed via the MintMT keyword `BUSEVENTINFO(1)`

These features can allow us to make our MintMT program event-driven and take action to execute certain code sections only when necessary (rather than polling the panel continuously).

Often, on controllers that support multitasking, the Bus event is used to start a Task that handles the new HMI data. This has the advantage of clearing the Event quickly so that multitasking can continue and hence the main program resumes promptly.

Note: On Flex+Drive^{II}, which does not support multitasking, each case would have to be handled in the event directly, during which time the main program is suspended.

The multitasking approach is illustrated by the following code:

```
Event BUS1
  Dim nEvent As Integer
  Dim nInfo As Integer
  nEvent = BUSEVENT(1)
  nInfo = BUSEVENTINFO(1)

  Select Case nEvent
    Case _bethMI_COMMS_UPDATE      `User changed data on HMI

      Select Case nInfo
        Case 255                  `User changed Product Length
          Run NewLength
          `Add cases for other adjustable values here
          `e.g. Case 256 may be a new Product Width
        End Select

      Case Else                    `Other Can event such as Node live
      End Select
  End Event

Task NewLength
  fProductLength = COMMS(2,255)
  `As an example, code in the rest of this task may calculate new move
  `parameters based on the new value of Product Length entered by the operator
End Task
```

Mint^{Multi-Tasking} MT Application Note

Recipe Download

When a Recipe is 'downloaded' by a CANopen HMI the data transfer actually takes place between the HMI's recipe data area and its working integer and floating point Comms areas (i.e. no data is actually transferred at this point in time to the Mint controller).

The HMI automatically generates a bus event of type 12 (pre-defined Mint constant `_betHMI_COMMS_UPDATE`) for each Recipe element that is downloaded. This in turn results in a 'queue' of BUS1 events being generated at the Mint controller.

If the BUS1 event is coded as in the example shown previously then this doesn't cause an issue as details of which specific COMMS location has changed are stored in variable `nInfo` and this persists for the duration of the event (i.e. its value cannot change until the current event completes and the next pending BUS1 event is processed).

Now consider the following Mint code example:

```
Event BUS1
  nEvent = BUSEVENT(1)
  nInfo = BUSEVENTINFO(1)

  Select Case nEvent

    Case _betHMI_COMMS_UPDATE
      If TaskStatus(HmiRead) = _tskTERMINATED Then Run HmiRead

    Case Else
      'Do nothing

  End Select

End Event

Task HmiRead
  Select Case nInfo

    Case 255
      fProductLength = COMMS (2, 255)

  End Select

End Task
```

At first glance it appears that this code will achieve the same result as our first example, albeit with the code spending less time executing the BUS1 event (which could be an advantage to the application).

However, if we consider what happens when a recipe is downloaded by the HMI then we can see that any BUS1 events that occur whilst the task 'HmiRead' is executing (e.g. once the first BUS1 event occurs) will effectively be 'thrown away' by the Mint program (and therefore local variables may not be updated according to new values stored in the HMI's recipe). Additionally it is possible that the global variable `nInfo` could be updated by a subsequent event causing the HmiRead task to process a CASE that didn't correspond with the original (first) BUS1 event.

Mint^{Multi-Tasking} MT Application Note

The problem of subsequent BUS1 events being discarded cannot be solved by running the HMIRead task every time an event occurs (because the task may not have completed processing the information supplied by the previous event).

We therefore need some form of 'buffer' in which subsequent event information can be stored ready for processing by the HMIRead task once it has completed execution.

In Mint we will create a circular buffer (i.e. an array of data). This buffer needs two 'pointers' to indicate which is the latest data (the head pointer) and which is the data being processed by the HMIRead task (the tail pointer). The program can determine that all available data has been processed when the tail pointer matches the head pointer.

The example code below illustrates this scheme:

```
`Global variables...
Dim nEventInfoBuffer(0 To 99) As Integer
Dim nEvent As Integer
Dim nInfo As Integer
`Constants...
Const nCircBufferSize As Integer = 100

Event BUS1
  nEvent = BUSEVENT(1)
  nInfo = BUSEVENTINFO(1)

  Select Case nEvent

    Case _betHMI_COMMS_UPDATE
      nHead = (nHead + 1) % nCircBufferSize
      nEventInfoBuffer(nHead) = nInfo
      If TaskStatus(HmiRead) = _tskTERMINATED Then Run HmiRead

    Case Else
      'Do nothing

  End Select

End Event

Task HmiRead
  Repeat
    nTail = (nTail + 1) % nCircBufferSize
    Select Case nInfo

      Case 255
        fProductLength = COMMS(2, 255)

    End Select
  Until nTail = nHead
End Task
```

Mint^{Multi-Tasking} MT Application Note

To ensure that this scheme is successful all we need to do is set the size of the circular buffer (defined by the constant `nCircBufferSize`) to be slightly larger than the maximum number of recipe elements that can be downloaded in one operation.