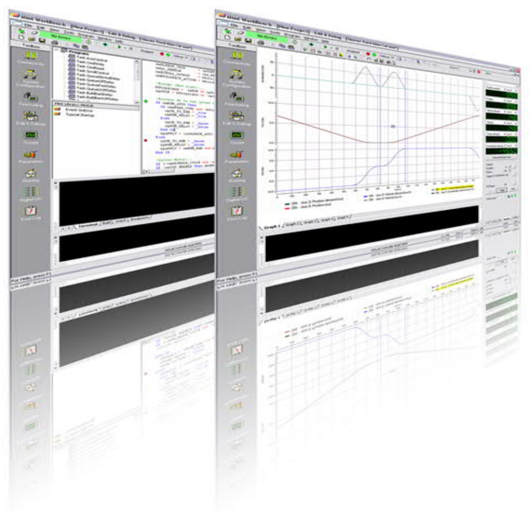
Motion Control Products

Application note
Using time and timers in Mint

AN00214

Rev B (EN)

Use built-in time features within Mint or create your own timer objects to suit your application.



Introduction

In most applications there are usually requirements to achieve one or more of the following functions:

- Wait (for a defined time) before continuing with program execution
- Perform an action at a regular time interval
- Time a process
- Perform an action when a condition has been met for a defined period of time
- Perform an action when a condition is not met for a defined period of time

All of these are easily achieved with the Mint programming language. Mint provides a variety of easy-to-use time related functions and keywords which we will detail in this application note. In addition, it is possible (via the use of Structure) to create your own timer objects (e.g. to create timer objects that function much like IEC61131 timers). An example of this is also provided with this application note.

Wait

Mint is a sequential language (as opposed to a cyclic PLC system) so execution of a line of Mint in a code module does not occur until the previous line has completed. This makes Mint ideal for programming machine sequences as there is no need to create a complex "machine state diagram" to handle each element of the sequence as the cycle is performed. Of course, Mint is so flexible it's still possible to create a looping task with an internal state diagram if you want to!

At some point in the sequence you may need to wait for a specified amount of time (e.g. maybe to give a pneumatic cylinder time to extend). Mint provides the WAIT instruction for this, taking a time in ms as a parameter.

Example:
Turn on output 0, wait 250ms, then move axis 2 100 user units…

```
OUTX(0) = _on
WAIT(250)   'suspend code in this module for 250ms
MOVER(2) = 100 : GO(2)
```

There is no limit to the number of WAIT instructions that can be used and these can be distributed throughout the program as required.

Perform an action at a regular time interval

Mint provides a single timer event. The interval for this event is set using the TIMEREVENT keyword. This keyword takes a time (in ms) as a value.

Example:
Use a timer event to calculate machine speed (in m/min) based on change in a wrapped encoder value (scaled in mm) every second…

```
'Main program
Dim fEncoderSnapshot
Dim fLastEncoderSnapshot
Dim fMachineSpeed

TIMEREVENT = 1000

Loop
 ….
End Loop

Event TIMER
  fEncoderSnapShot = ENCODER(0)
  fMachineSpeed = 0.06 * WrapOffset(fLastEncoderSnapshot,fEncoderSnapShot,0,ENCODERWRAP(0))
  fLastEncoderSnapShot = fEncoderSnapShot
End Event
```

To disable the timer event set TIMEREVENT = 0.

Time a process

The Mint main program (parent task) and every Mint child task have their own record of time (in ms) which is accessed via the TIME keyword. So for example, TIME in one task can be set to 0 and then queried/read when needed and this is completely independent to TIME in a second task (which may also be reset and queried by the logic within that task).

Each task can therefore independently time a process via the TIME keyword.

Example:
Time how long it takes a pneumatic cylinder to extend (e.g. turn on an output and time how long it takes the input to come on)

```
Dim nExtendTime As Integer
TIME = 0
OUTX(1) = _on
PAUSE INX(4)
nExtendTime = TIME
```

For controllers running firmware that supports compiler target format 14 onwards (e.g. NextMove ESB-2 using 5454 onwards, e100, e180 and e190 products) it is also possible to declare variables of type 'Time'.

e.g. Dim tTimer1 As Time

These variables can then be set to a value (e.g. reset to 0) and queried by the program (with their scope depending on whereabouts in the program the variable was declared). The only limit to the number of time variables that can be used is available program space.

Perform an action when a condition has been met for a defined period of time

There are a number of different ways to achieve this function. One example might be to use a loop structure of some kind…

Example:
Continue program execution when input 5 has been on for 3 seconds continuously…

```
Dim tConditionTimer As Time

tConditionTimer = 0
Repeat
  If INX(5) = _off Then tConditionTimer = 0
Until tConditionTimer >= 3000
```

This example is fine for situations where you want to "suspend" the code until a condition has been true for a period of time but maybe this condition needs to be monitored "in parallel" with the execution of the rest of the program…..in this case make use of a Mint task.

Example:
Main program runs a background task that in turn detects that input 5 has been held on for 3 seconds and then turns on output 1...

```
'Main program…
Run (ConditionMonitor)
….

Task ConditionMonitor
  Dim tConditionTimer As Time
  tConditionTimer = 0
  Repeat
    If INX(5) = _off Then tConditionTimer = 0
 Until tConditionTimer >= 3000
 OUTX(1) = _on
End Task
```

Code to take action when a condition is not met (false) for a certain time is very similar, we just need to invert the logic for resetting the timer.

Creating IEC style timer functions

Using the Mint STRUCTURE keyword we can create our own timer objects…

```
Structure T_IECTimer
 IN As Integer
 PT As Integer
 ET As Time
 Q As Integer
End Structure
```

Here we have reused the IEC definition for the input and output parameter names. It is not ideal that IN clashes with the Mint keyword to read an input bank, but in this case it is not a problem as the compiler will recognise that .IN is a member variable of the structure (the editor will however change the capitalisation and colour of the text as if it were being used as the Mint keyword).

For this example we will create a TON (delay on) timer – we'll simulate a sensor looking for a continual queue of products. If the queue is present for a preset time (2 seconds for this example) then we will set the timer output (which may then be used to start a machine process for example).
First we must declare a variable of our new type…

```
Dim tQueue As T_IECTimer
```

Now we need some logic to start/stop our timer…..our “product sensor” is wired to input 6, so in our program we have configured input 6 to be triggered on a rising and trailing edge (see INPUTMODE, INPUTPOSTRIGGER and INPUTNEGTRIGGER in the Mint Help file for further details on input configuration if needed).
As a result we can use the Mint Event IN6 to detect when the product sensor turns on or off…when the input turns on we will load our preset time (2 seconds), initially clear the output and enable our timer. If the input turns off we will disable the timer and turn off the timer output and reset the elapsed time…

```
Event IN6
  If INSTATEX(6) Then
    ‘Rising edge…
    tQueue.IN = _On
    tQueue.PT = 2000
    tQueue.Q = _Off
    tQueue.ET = 0
  Else
    ‘Trailing edge…
    tQueue.IN = _Off
    tQueue.Q = _Off
    tQueue.ET = 0
  End If
End Event
```

We can make this code more “re-usable/modular” by creating a subroutine for initialising the timer and passing parameters to this routine. We can then use this routine to enable/disable any timers that we might include in our program…

```
Sub doInitTONTimer (ByRef tTimer As T_IECTimer, ByVal nPreset As Integer, ByVal bEnable As Integer)
  If bEnable Then
    tTimer.IN = _On
    tTimer.PT = nPreset
    tTimer.Q = _Off
    tTimer.ET = 0
  Else
    tTimer.IN = _Off
    tTimer.Q = _Off
    tTimer.ET = 0
  End If
End Sub
```

Our input event can now be written as…

```
Event IN6
  If INSTATEX(6) Then
    ‘Rising edge…
    doInitTONTimer (tQueue, 2000, _On)
  Else
    ‘Trailing edge…
    doInitTONTimer (tQueue, 2000, _Off)
  End If
End Event
```

In our program logic (e.g. in a Mint task that is cycling all the time performing background logical tests, much like a PLC cyclic task processing general logic) we may now want to check if our timer has completed. Again, to make the code modular and reusable we will create a function to test if the timer output should be turned on. The function will return the state of a logical

‘AND’ of the timer output and the enable input. Additionally, in case some program logic other than our input event has disabled the timer, we can use the function to disable the timer…

```
Function doTON (ByRef tTimer As T_IECTimer) As Integer
  If tTimer.IN Then
    ‘Timer is still enabled so check whether on time has elapsed…
    If tTimer.ET >= tTimer.PT Then tTimer.Q = _On
  Else
    ‘Timer has been disabled so turn it off…
    tTimer.ET = 0
    tTimer.Q = _Off
  End If
  doTON = (tTimer.Q AndAlso tTimer.IN)
End Function
```

We would use this function in our cycling Mint task as follows…

```
Task PLCLogic
  Loop
    If doTON(tQueue) And TaskStatus(StartMachine) = _tskTerminated Then Run(StartMachine)
    ….
  End Loop
End Task
```

If we want to check timer properties such as the elapsed time these can be directly accessed from any part of the program as and when required…

```
e.g. nElapsedTime = tQueue.ET
```

Logic for a TOF (delay off) style timer is very similar; please refer to the Mint program example included with this application note for a more detailed example. The Mint program example is designed to allow the code to be tested with the Mint Virtual Controller included as part of Mint Workbench and as such uses a NETDATA event rather than input events for input simulation. Use the Data Watch window or Command window to write 0 or 1 to NETINTEGER(0) to test the operation of the TON and TOF timers. The preset times are set to 5 seconds for the TON and 3 seconds for the TOF. This allows time for the user to change the Netdata value. Add the IEC timer variables to the variable watch window to observe the operation of the timer outputs (Q) or watch the status of the ‘StartMachine’ task. The task will not start until NETINTEGER(0) is > 0 for more than 5 seconds and will not terminate until NETINTEGER(0) has been turned off continuously for at least 3 seconds.

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