

**Mint<sup>MT</sup>** Multi-Tasking  
**Application Note****AN00117-001 - Using the Move Buffer****Related Applications or Terminology**

- **Synchronised Motion**
- **Interpolated Motion**
- **Contouring**
- **Feedrate control**
- **Blending**

**Supported Controllers**

<b>NextMove</b> PCI	<input checked="" type="checkbox"/>
<b>NextMove</b> BX <sup>II</sup>	<input checked="" type="checkbox"/>
<b>NextMove</b> ST	<input checked="" type="checkbox"/>
<b>NextMove</b> ES	<input checked="" type="checkbox"/>
<b>NextMove</b> ESB	<input checked="" type="checkbox"/>
<b>NextMove</b> 100	<input checked="" type="checkbox"/>
<b>MintDrive</b> <sup>II</sup>	<input checked="" type="checkbox"/>
<b>Flex+Drive</b> <sup>II</sup>	<input checked="" type="checkbox"/>

**Overview**

One of the powerful features of MintMT is the move buffer. The move buffer allows multiple moves to be loaded in advance and started on demand. It allows many axes to be synchronised and complex interpolated move types to be performed. Moves can be contoured together to make smooth paths, and feedrate control allows these moves to be performed with differing speeds and accelerations. Moves can also be blended together in order to reduce motion time. The move buffer can even inform a MintMT program when it needs restocking.

**Loading and Triggering Moves**

When a move command is issued, the move is stored and executed from the move buffer. To start the move off, called triggering, the **GO** keyword is used. For example:

```
MOVER(0) = 10      ' Load a relative move of 10 units
GO(0)             ' Trigger the motion
```

For some move types, it is possible to load more than one move at once. For example:

```
MOVER(0) = 10      ' Load a relative move of 10 units
MOVER(0) = 20      ' Load a relative move of 20 units
GO(0)             ' Trigger the motion
```

Here, two moves are loaded into the buffer and then triggered. The **GO** keyword will trigger all moves in the move buffer so it only needs to be called once. The axis will move forward 10 units and stop. The axis will then move forward 20 units.

The **GO** keyword can also be used to synchronise motion on multiple axes. For example, the following code will start motion on axes 0 and 1 simultaneously:

```
MOVER(0) = 10      ' Load a relative move of 10 units on axis 0
MOVER(1) = 20      ' Load a relative move of 20 units on axis 1
GO(0,1)           ' Trigger the motion on axes 0 and 1
```

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

Sometimes it is inconvenient to trigger every move as it is loaded into the move buffer. The **TRIGGERMODE** keyword can be used to enable automatic triggering. This means that a move begins to execute as soon as it is loaded into the move buffer.

```
TRIGGERMODE(0) = 0           ' Enable automatic triggering
MOVER(0) = 20                ' Load and start a relative move of 20 units
```

## Sizing the Move Buffer

Each axis has its own move buffer which can be set to a user defined size. For some move types, many moves can be loaded in advance, for others, only one move can be loaded. MintMT will only halt program execution when it reaches a command that affects the move buffer, such as trying to load a move when the move buffer is full. It is important to make sure that your code will always trigger motion before the move buffer becomes full otherwise program execution will halt. The following code segment shows this behaviour:

### Insufficient move buffer size

```
MOVEBUFFERSIZE(0) = 2
MOVEA(0) = 10
MOVEA(0) = 20
MOVEA(0) = 30 ← Execution would halt here
GO(0)
```

### Sufficient move buffer size

```
MOVEBUFFERSIZE(0) = 3
MOVEA(0) = 10
MOVEA(0) = 20
MOVEA(0) = 30
GO(0)
```

Where the move buffer size is set to two, after two moves have been loaded, program execution will pause (within the task) until a space in the buffer becomes available to load the third move. Where the move buffer size is set to three, the program flow is not halted.

A large move buffer size is required when moves are to be contoured together, so that the axes do not start and stop between each move. Increasing the move buffer size does increase memory usage and will slow MintMT slightly. The move buffer size should be set in the **Startup** block of the MintMT program before any axes are enabled.

## Restocking the Move Buffer

Where moves are repeatedly loaded, it may not be practical to allocate the move buffer large enough to contain all the moves. The number of free spaces in the buffer can be read with the **MOVEBUFFERFREE** keyword. In the following example, moves are loaded using the Comms array. Position is written out to a keypad so program execution cannot be allowed to stop.

```
Loop
' Wait until data valid flag (Comms location 1) and there is space in the buffer
If (COMMS(1) = 1) And (MOVEBUFFERFREE(0) > 0) Then
  MOVER(0) = COMMS(2)           ' Load move with data (Comms location 2)
  GO(0)                         ' Trigger motion
  COMMS(1) = 0                 ' Indicate to host that move is loaded
EndIf
Print #3, "Pos = ", POS(0)     ' Keypad is assigned to terminal 3
End Loop
```

# MintMT<sup>TM</sup> Multi-Tasking Application Note

## Automatically Restocking the Move Buffer (NextMove only)

To lessen the work involved in checking and keeping the move buffer stocked with moves, MintMT can generate an event when the number of free spaces in the move buffer reaches a threshold. Moves can be loaded from the event handler and then program execution will resume as normal.

```
'Main code block
Loop
...
End Loop

Event MOVEBUFFERLOW
  ' Wait until data valid flag (Comms location 1)
  If (COMMS(1) = 1) Then
    MOVER(0) = COMMS(2)      ' Load move with data (Comms location 2)
    GO(0)                   ' Trigger motion
    COMMS(1) = 0           ' Indicate to host that move is loaded
  EndIf
End Event

' Startup code - executed when the controller is powered up
Startup
  ' Setup the move buffer
  MOVEBUFFERSIZE(0) = 20    ' Create 10 spaces in the move buffer
  MOVEBUFFERLOW(0) = 5     ' Create event when there are more than 5 spaces
End Startup
```

## Keeping Track of Moves (NextMove only)

To keep track of moves in the buffer, it is possible to attach an identifier to each move as it is loaded and to read the identifier of the currently executing move. For example:

```
MOVEBUFFERSIZE(0) = 20      ' Create 20 spaces in the move buffer
MOVEBUFFERID(0) = 1        ' Set the ID to be stored in the buffer to 1
MOVER(0) = 20              ' Load and start a relative move of 20 units
MOVEBUFFERID(0) = 2        ' Set the ID to be stored in the buffer to 2
MOVER(0) = 10              ' Load and start a relative move of 10 units
GO(0)                     ' Trigger motion
Pause MOVEBUFFERID(0) = 2  ' Wait until the second move starts to execute
OUTX(0) = _ON              ' Turn on digital output 0
```

Once motion has finished, it is possible to read the ID of the last move executed using the **MOVEBUFFERIDLAST** keyword.

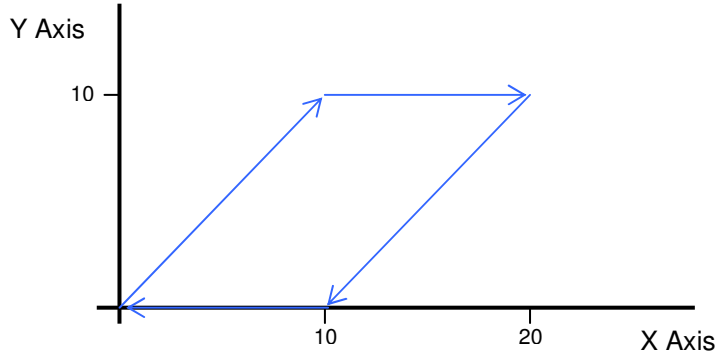
## Interpolated (multi - axis) Motion (NextMove only)

NextMove can perform the following multi-axis interpolated moves: **VECTORA/VECTOORR**, **CIRCLEA/CIRCLER** and **HELIXA/HELIXR**. When a multi-axis move is loaded, the first axis specified is known as the master axis. The **SPEED**, **ACCEL**, **DECEL**, **ACCELJERK** and **DECELJERK** parameters set for the master axis are used for the motion path of the interpolated move. It is important to set the move buffer size for all of the axes involved in a multi-axis move.

# Mint MT Application Note

```
MOVEBUFFERSIZE([0,1]) = 4;
VECTORA(0,1) = 10, 10
VECTORA(0,1) = 20, 10
VECTORA(0,1) = 10, 0
VECTORA(0,1) = 0, 0
GO[0,1]
```

- ' Create 4 spaces in the move buffer for axis 0 and 1
- ' Load an absolute vector move to position 10, 10
- ' Load an absolute vector move to position 20, 10
- ' Load an absolute vector move to position 10, 0
- ' Load an absolute vector move to position 0, 0
- ' Trigger motion

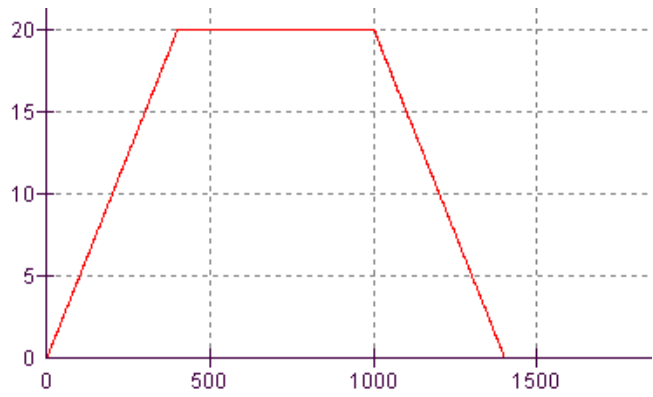
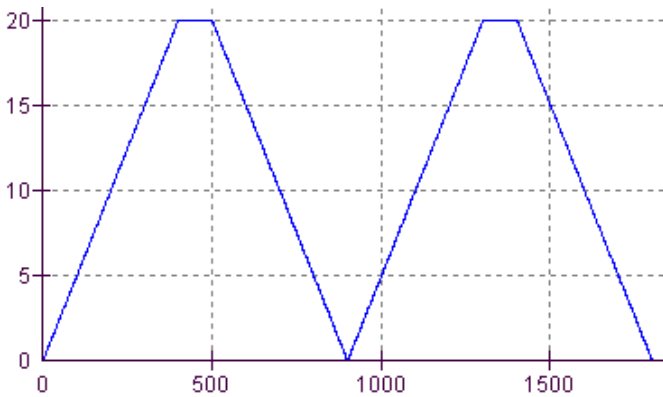


## Creating Contoured Paths (NextMove only)

In a contoured move a series of moves are joined together so that they are performed with a constant path velocity. For example, the following plots show the effect of contouring:

```
CONTOURMODE(0) = _ctmCONTOUR_OFF
MOVER(0) = 10
MOVER(0) = 10
GO(0)
```

```
CONTOURMODE(0) = _ctmCONTOUR_ON
MOVER(0) = 10
MOVER(0) = 10
GO(0)
```

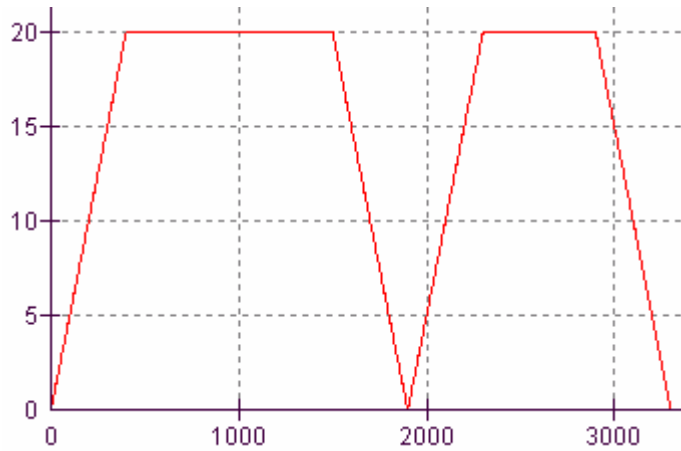


When a move is loaded into the buffer, the contour mode is stored as well, allowing some moves to be loaded with contouring turned on and others with it turned off. For example:

# Mint MT Application Note

```

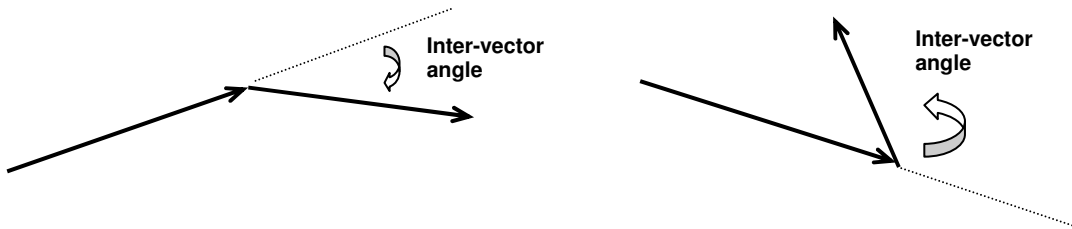
MOVEBUFFERSIZE(0) = 5           ' Create 5 spaces in the move buffer
CONTOURMODE(0) = _ctmCONTOUR_ON ' Turn contouring on
MOVER(0) = 10                   ' Load a relative move of 10 units
MOVER(0) = 10                   ' Load a relative move of 10 units
CONTOURMODE(0) = _ctmCONTOUR_OFF ' Turn contouring off
MOVER(0) = 10                   ' Load a relative move of 10 units
CONTOURMODE(0) = _ctmCONTOUR_ON ' Turn contouring on
MOVER(0) = 10                   ' Load a relative move of 10 units
MOVER(0) = 10                   ' Load a relative move of 10 units
GO(0)                            ' Trigger motion
    
```



When a move is loaded, the state of contouring controls whether the end of that move will be a stop point. In the example above, the third move was loaded with contouring off, making the end of that move a stop point.

When interpolated moves are loaded with contouring turned on, the path speed is kept constant across multiple vectors. This means that the axis components of a move may not be continuous in velocity. There will be a 'clunk' at vector boundaries. To avoid jarring the machine, the angle between vectors should be kept small. MintMT can automatically look out for sharp angle changes using inter-vector angle control.

The inter-vector angle is defined as the angle between two vectors or arcs.

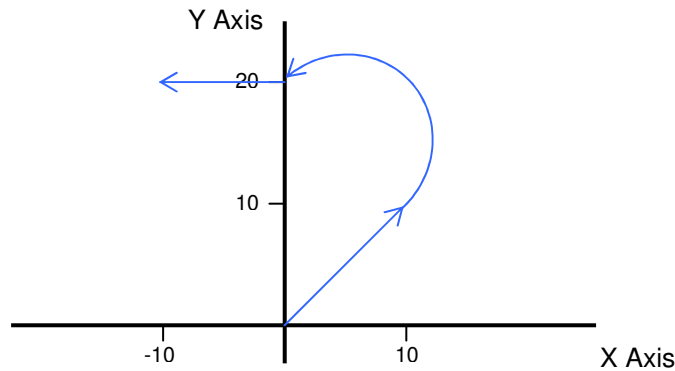


If the angle between two vectors exceeds the threshold set with **CONTOURPARAMETER.axis.0**, then a stop point will be created, otherwise the axes will continue to give a constant path speed.

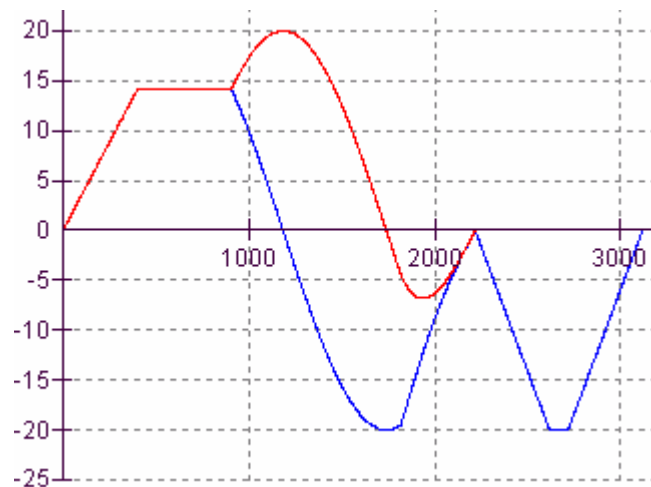
# Mint MT Application Note

```

MOVEBUFFERSIZE([0,1]) = 5;           ' Create 5 spaces in the move buffer
CONTOURMODE([0,1]) = _ctmCONTOUR_ON + _ctmVECTOR_ANGLE_ON;
                                     ' Turn contouring on with inter-vector angle control
CONTOURPARAMETER([0,1],0) = 10;     ' Set threshold to 10 degrees.
VECTRR(0,1) = 10, 10                 ' Load a relative move of 10 units
CIRCLR(0,1) = -5, 5, 180             ' Load a relative move of 10 units
VECTRR(0,1) = -10, 0                ' Load a relative move of 10 units
GO(0,1)                              ' Trigger motion
    
```



The axes will automatically stop between the arc and the second vector since the angle between these two vectors exceeds the threshold of 10 degrees. The velocity traces of X and Y are shown below:



When performing a series of contoured moves, it is important to maintain as much distance as possible in the move buffer. If the move distance remaining in the move buffer ever drops to less than the distance required to decelerate the axis to a stop, then the axes will start to decelerate. Once started, the axes will come to a stop before starting again, even if more moves are loaded into the buffer.

The **SUSPEND** keyword can be used to pause motion in the move buffer as well as to single step through individual moves in the move buffer.

**Mint<sup>MT</sup> Multi-Tasking** Application Note**Feedrate Control (NextMove only)**

As moves are being executed from the move buffer, if any of the motion profile parameters are changed (**SPEED/FEEDRATE**, **ACCEL**, **DECEL**, **ACCELJERK** and **DECELJERK**), the change will have an immediate affect on the move in progress. Feedrate control allows these profile parameters to be loaded into the move buffer with each move. For example, in a routing application, some moves may be required to move at a different speed. The following example moves around a square with rounded corners. The corner sections are performed at a slower speed than the straights:

```
MOVEBUFFERSIZE([0,1]) = 10;           ' Create 10 spaces in the move buffer
CONTOURMODE([0,1]) = _ctmCONTOUR_ON + _ctmVECTOR_ANGLE_ON;
CONTOURPARAMETER([0,1],0) = 10;       ' Turn contouring on with inter-vector angle control
FEEDRATEMODE([0,1]) = _frFEEDRATE;    ' Set threshold to 10 degrees.
FEEDRATE([0,1]) = 20;                 ' Store feedrate in the move buffer
VECTORA(0,1) = 20, 0                  ' Set the feedrate to 20 units/s
FEEDRATE([0,1]) = 10;                 ' Set the feedrate to 10 units/s
CIRCLEA(0,1) = 20, 5, 90
FEEDRATE([0,1]) = 20;                 ' Set the feedrate to 20 units/s
VECTORA(0,1) = 25, 25
FEEDRATE([0,1]) = 10;                 ' Set the feedrate to 10 units/s
CIRCLEA[0,1] = 20, 25, 90
FEEDRATE([0,1]) = 20;                 ' Set the feedrate to 20 units/s
VECTORA(0,1) = 5, 30
FEEDRATE([0,1]) = 10;                 ' Set the feedrate to 10 units/s
CIRCLEA(0,1) = 5, 25, 90
FEEDRATE([0,1]) = 20;                 ' Set the feedrate to 20 units/s
VECTORA(0,1) = 0, 5
FEEDRATE([0,1]) = 10;                 ' Set the feedrate to 10 units/s
CIRCLEA(0,1) = 5, 5, 90
GO(0,1)                                ' Trigger motion
```

In this example, **FEEDRATEMODE** is set to **\_frFEEDRATE** which means store feedrate in the move buffer. Other bits in the **FEEDRATEMODE** keyword allow **ACCEL** and **DECEL** and **ACCELJERK** and **DECELJERK** to be stored in the move buffer.

The **FEEDRATEMODE** keyword also allows some moves to be loaded as 'fast traverse' moves. These moves will respond to the **FEEDRATEOVERRIDE** keyword which allows the speed of fast traverse moves to be modified, even though the feedrate of that move was specified when the move was loaded.





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

## Output Moves (NextMove only)

It is sometimes useful to be able to change the state of digital outputs when a move starts to execute. MintMT allows digital output moves to be loaded into the move buffer so that they are executed with the motion. For example:

```
MOVEBUFFERSIZE(0) = 10      ' Create 10 spaces in the move buffer
MOVER(0) = 10               ' Relative move of 10 units
MOVEOUT(0,0) = 7           ' Turn on output channels 0, 1 and 2 on bank 0
MOVER(0) = 10               ' Relative move of 10 units
MOVEOUTX(0,1) = 0          ' Turn off output channel 1
MOVER(0) = 10               ' Relative move of 10 units
MOVEPULSEOUTX(0,3) = 100   ' Turn on output channel 3 for 100ms
MOVER(0) = 10               ' Relative move of 10 units
GO(0)                       ' Trigger motion
```

At the end of the first move, digital outputs 0, 1 and 2 on output bank 0 are activated and all other outputs on bank 0 are deactivated. At the end of the second move, digital output channel 1 is deactivated. At the end of the third move, digital output channel 3 is activated and a timer started. The fourth move is then started. After 100ms, digital output channel 3 is deactivated.