Motion Control Products

# Application note
# Stop handler framework

AN00190

Rev D (EN)

Pre-written code sample illustrates a simple method of handling the Mint stop event whilst still allowing other program tasks to continue executing

## Introduction

Mint is a sequential language (i.e. a particular line of code is not executed until the previous line has completed execution) that also provides the user with the ability for parallel processing (via multi-tasking) and event/interrupt driven operation (e.g. via digital input events) making the language, as a whole, incredibly flexible and particularly suited to machine and motion control applications.

Very often an application requires the ability to keep processing certain elements of code (e.g. HMI inputs/outputs, general machine logic) even whilst in an Emergency stop condition. This is easy to achieve using Mint and this application note sets out to describe how this is possible and how a basic Mint code framework can be re-used in most applications. This framework is provided in the form of a Mint code sample, suitable for NextMove e100, MicroFlex e190 and MotiFlex e180 controllers, which accompanies this application note. It is also available to be dragged from the Mint code library included as part of Mint Workbench.

Please contact your local ABB motion support team if using an older (classic or legacy) motion product.

## Framework requirements

The overall requirements for the Mint framework are as follows:

- To respond to the stop input event
- To stop motion on axes where applicable
- To ensure the stop input event is not called again until the stop input is cleared and then reactivated
- To ensure no further motion is possible until the stop input has cleared

We will detail how each of these requirements are handled by the Mint sample code in the following sections.

## Mint stop event

The Mint STOPINPUT keyword allows the user to configure which digital input is used for connection of the emergency stop signal from the machine. Typically this input is active low (i.e. considered active when the input is not connected) so as to be ‘fail-safe’. As a result the corresponding bit in INPUTACTIVELEVEL is set to zero.

The Mint keyword STOPMODE can be used to configure the action taken by the firmware as a result of the stop input becoming active.

In all cases the controller will call 'Event Stop' if it exists.
It is common to use a constant to define the input number being used as the stop input (to make the code both more readable and to allow this input to be changed at a later date with minimal changes to the code).

In the case of our sample program (designed to run on the virtual controller) we have allocated digital input channel 0 as the Mint stop input.

The following sections of code relate to configuration of the stop input…

```
'Setup input 0 as the stop input channel….
Const _nStopInput As Integer = 0

'Assign the relevant input as the stop input…
STOPINPUT(0) = _nStopInput

'Setup the controller to stop the axis rapidly when the stop input activates…
STOPMODE(0) = _smCRASH_STOP  (or STOPINPUTMODE(0) = _siCANCEL for non-e100 controllers)

'Configure the relevant input to be active low…
INPUTACTIVELEVEL(0) = 01111111111111111110
```

(Note that in this case there is no need to configure input 0 to be edge triggered).

```
'Define a stop event handler that will be called when the stop input activates…
Event Stop
  doHandleStop
End Event
```

The code to execute when the input event occurs is placed in a subroutine for convenience (should we need to re-use this logic for any reason).

The doHandleStop subroutine takes the following actions…

- Sets a global variable (bStopActive) which is used throughout the application to determine if a stop input is active. This variable can then be used to prevent sections of code from executing (e.g. prevent motion)
- Cancels motion (this is optional as the STOPMODE/STOPINPUTMODE action will have taken place anyway)
- Waits for the axis to become IDLE (i.e. motion has come to a halt)
- Calls a subroutine (doEndTasks) responsible for ending all tasks that should perform no function in the event of an Emergency Stop being active (in our example this is just the task responsible for moving the axis but in practice this may not be limited just to tasks performing motion)
- Disables the axis
- De-assigns the Mint STOPINPUT (by setting it to -1). This ensures that the Mint stop event is not repeatedly called whilst the digital input originally defined as the STOPINPUT remains active
- Runs a task (StopHandler) responsible for determining when the Emergency Stop has been cleared

### StopHandler task

The Mint StopHandler task is started as a result of activation of the Mint stop event. This task now waits for the digital input associated with the emergency stop to deactivate. Once this input has deactivated the task can:

- Reassign the Mint STOPINPUT (so the Stop Event can now reoccur should the input activate)
- Clear the global variable (bStopActive) used to interlock sections of code

### Parent Task

The parent task assigns the STOPINPUT and initializes the state of bStopActive whenever it starts. This is to ensure the program initializes correctly if it is ever restarted by a Mint RUN command.

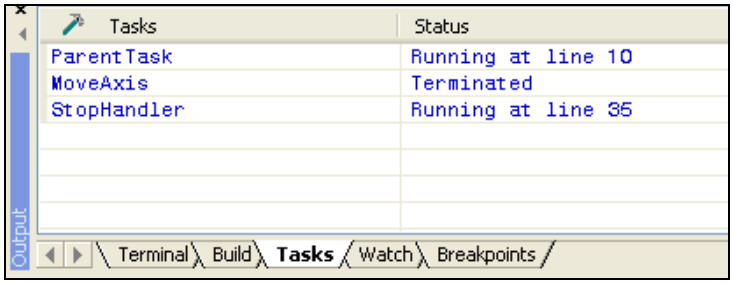
## Testing the example code

The example framework code is written for the virtual controller and contains elements to allow motion to be initiated on a single axis (axis 0). The example code is easily adapted for use in any application (and for any number of axes).

Start Workbench, start a new project and select the virtual controller.
From the File>Open File… menu select the example program (it will appear in the Editor window).

Select Program>Compile, Download and Run to download this file to the virtual controller.

In the example program input 0 has been defined as the stop Input (and by default it is active low). As there is no wiring to the virtual controller, the stop Input will initially be active. The 'Tasks' watch window should show that the Parent Task and StopHandler tasks are running…




If you enter the variable ParentTask::bStopActive into the Watch window you will find this is set to 1.

Enter the following text at the Workbench command window (and press Return)…

**INPUTACTIVELEVEL(0) = INPUTACTIVELEVEL(0) or 1**

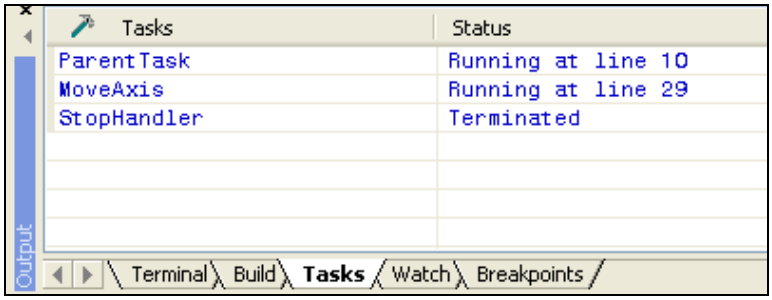
This sets input 0 to be active high (which in turn will de-activate the stop input). You will see from the Tasks watch window that the StopHandler task terminates and the variable watch window will show bStopActive is now set to 0.

Now the stop input is inactive enter the following text at the Workbench command window (and press Return)…

**COMMS(1) = 1**

Writing to Comms(1) from outside of the Mint program causes a Mint level Comms1 event to occur (if it exists). In our example program we use this event to start the MoveAxis task (if it isn't already running).

The Tasks watch window will show that MoveAxis is running and the Spy Window can be used to monitor changes in axis 0's position and speed as indexes occur…




Now enter the following text at the Workbench command window (and press Return)…

**INPUTACTIVELEVEL(0) = INPUTACTIVELEVEL(0) and 0xFFFFE**

This sets input 0 back to being active low. As a result the stop input will occur, motion will be stopped on axis 0, task MoveAxis will be terminated, task StopHandler will restart and variable bStopActive will be set to 1 (_True).

Try writing to Comms(1) at the command window again. You will find that no motion occurs. This is because the interlock at the beginning of task MoveAxis detects that the stop is active and ends the task straight away…

**If bStopActive = _true Then End MoveAxis**

This completes this application note. You should now be in a position to understand and re-use where relevant the example code provided.
If you wish to store this code in your own Mint Library (so you can simply drag it into the editor window in future) use the mouse/keyboard to select/highlight all the code in the editor and drag this to the Mint Library pane in the Editor window. Enter a suitable name for this code segment and it is now ready to be dragged out of the library and into Mint applications when required.

The samples library provided with Workbench contains the core framework already - named “Stop Event Handler (All)”

## 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**