Motion Control Products

Application note Generic drive interface

AN00204

Rev U (EN)

Ready to use PLC function blocks, combine with pre-written Mint application for simple control of e180 and e190 drives via any available network/fieldbus

Introduction

This application note details the ‘Generic drive interface’ Mint program for MicroFlex e190 and MotiFlex e180 drives – a simple drive control profile that can be accessed via any communication interface that provides access to the drives’ Netdata array (e.g. Modbus TCP, EtherCAT, Ethernet/IP or PROFINET). This allows PLCs (such as the ABB AC500 or eco-PLC) to supervise and control simple motion functions on one or more axes. Example ABB PLC programs (for which Automation Builder v1.1.2 or later is required) are also included which utilise ‘PLCopen motion control’ style function blocks.
This particular application note focuses on operation via Modbus TCP but the principle is identical for all available networks, please refer to the motion website for additional GDI application notes specific to other fieldbus networks (e.g. AN00222 for Ethernet/IP, AN00234 for EtherCAT, AN00251 for PROFINET). Please contact your local ABB support team if you need legacy support information regarding GDI applications on older ABB motion products such as MicroFlex e150.

The sample programs with this application note provide a mechanism for an ABB PLC to:

- Issue a home command
- Issue a find end stop command (home to a preset torque limit, firmware version 5868 onwards required)
- Issue a relative move
- Issue an absolute move
- Issue an incremental relative move (and optionally stop a programmed distance past a “fast-capture” position)
- Issue an incremental absolute move (and optionally stop a programmed distance past a “fast-capture” position)
- Setup an offset target for an incremental move (i.e. position the axis relative to a captured fast interrupt)
- Jog the axis
- Set the axis position
- Issue a speed reference
- Issue a torque reference
- Enable/disable the axis
- Enable/disable hardware limits
- Reset axis errors
- Perform a controlled stop or crash stop on the axis
- Gear the axis to a secondary encoder input
- Set speed, acceleration times, deceleration times and jerk times for all motion
- Control modulo or non-modulo axes

Power and productivity for a better world™

At the same time the PLC is able to monitor status information from the drive including:

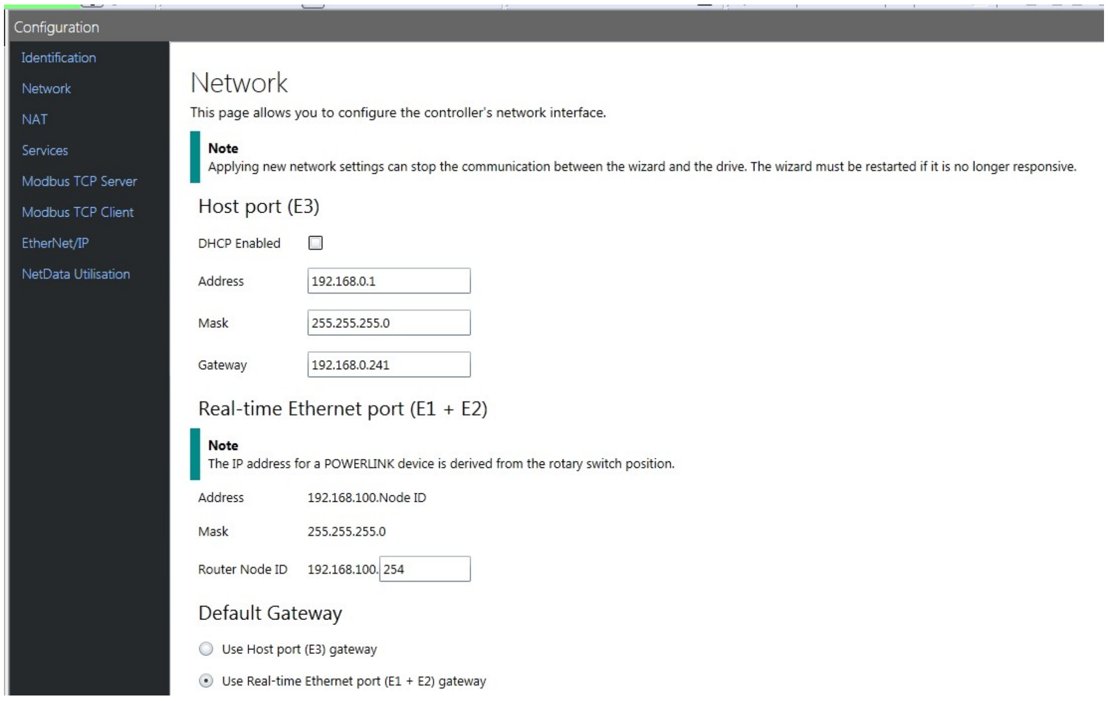
- Enabled state
- Ready to be enabled state
- Idle state
- In Position state
- Motor brake state
- Homed state
- Forward limit state
- Reverse limit state
- Fault state
- Stop input state
- Indication of missing fast latch interrupt
- Phase search status
- Error code
- Measured position
- Measured velocity
- Following error
- Axis mode of operation
- RMS current

This is all achieved via, what appears to the PLC as, Modbus registers. Because we have used 32 bit data for the interface each value utilizes two 16-bit Modbus registers which in turn are mapped onto a single 32-bit NETINTEGER or NETFLOAT location in the drive). An optional watchdog mechanism is also included, allowing the drive to take action (crash stop and disable by default) in the event of communication loss.

Communication configuration

MotiFlex e180 and MicroFlex e190 drives include native support for Modbus TCP. Mappings are automatically made between Modbus registers and Netdata. The Mint Workbench 'Configuration' section allows the user to configure the operating parameters for all of the available communication interfaces. The "Network" section lets the user set the drive's IP address for example…

MicroFlex e190 and MotiFlex e180 drives have a default IP address of 192.168.0.1 (when using firmware version 58xx which is recommended). This matches the PLC nicely as this has a default IP address of 192.168.0.10 (i.e. they are both on the same subnet by default).



Network

This page allows you to configure the controller's network interface.

Note
Applying new network settings can stop the communication between the wizard and the drive. The wizard must be restarted if it is no longer responsive.

Host port (E3)

DHCP Enabled ☐

Address	192.168.0.1
Mask	255.255.255.0
Gateway	192.168.0.241

Real-time Ethernet port (E1 + E2)

Note
The IP address for a POWERLINK device is derived from the rotary switch position.

Address	192.168.100.Node ID
Mask	255.255.255.0
Router Node ID	192.168.100.254

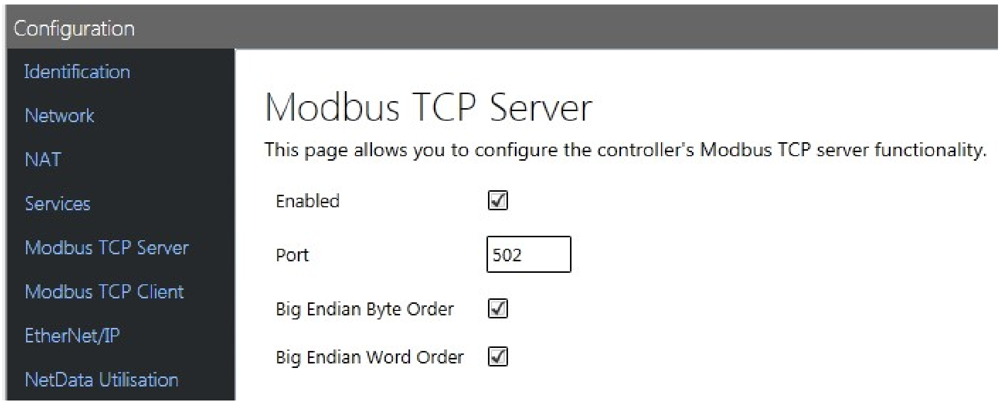
Default Gateway

○ Use Host port (E3) gateway

● Use Real-time Ethernet port (E1 + E2) gateway

The “Modbus Server” section of the Configuration pages allows adjustment of the following drive communication parameters relating to Modbus TCP:

- Port number (e.g. 502 is the standard port used for Modbus TCP)
- Byte Order (e.g. Big endian selected for use with AC500 PLC)
- Word Order (e.g. Big endian selected for use with AC500 PLC)



Modbus TCP Server

This page allows you to configure the controller's Modbus TCP server functionality.

Enabled	✓
Port	502
Big Endian Byte Order	✓
Big Endian Word Order	✓

Configuring the Generic Drive Interface (GDI) Mint program

The pre-written GDI Mint program only requires only a small amount of customisation to suit the user’s application. At the beginning of the main program (after the program header) are a set of application related constants…

```
' Application specific (edit as required)...
Const _bRemoteEnable As Integer    = _true   'can PLC control enable?
Const _bUseWatchdog As Integer     = _true
Const _nWatchdogTime As Integer    = 2000    'ms
Const _fScale As Float             = 131072    'counts per user unit
Const _nHomeType As Integer        = _hmNEGATIVE_SWITCH 'modify to suit home type
Const _nHomeInput As Integer       = 1       '
Const _nFwdLimit As Integer        = -1      'digital input
Const _nRevLimit As Integer        = -1      'number
Const _nLimitMode As Integer       = _emCRASH_STOP_DISABLE
Const _nStopInput As Integer       = -1      '
Const _nStopMode As Integer        = _smDECEL
#If _platform = _nMotiFlexE180 Then
  Const _nInputLevel As Integer    = 2#11111111    '0=Active low, 1=Active high
#Else
  Const _nInputLevel As Integer    = 2#1111
#End If
Const _nEncoderWrap As Integer     = 0       'Set to counts in 1 cycle for modulo axis
Const _fFolErrorFatal As Float     = 1       'User units - edit to suit application
Const _nMasterChannel As Integer   = 2       '1 = Fast inputs, 2 = Secondary encoder input
Const _nMasterEncMode As Integer   = 0       'Set to 0 or 1 as required for encoder direction
Const _nFollowMode As Integer      = _fmNO_RAMP  'Edit to use required Mint follow mode _fmXXXX
Const _nMotorDirection As Integer = 0       'Set to 0 or 1 as required for correct motor direction
```

Constant	Function
_bRemoteEnable	The user can decide whether the PLC has any control of the drive's enabled status or not
_bUseWatchdog	The user can decide whether the drive uses a watchdog mechanism to detect loss of communication from the PLC or not (if set _true then the drive will crash stop and disable in the event of loss of communication)
_nWatchdogTime	The user can set an appropriate timeout value (in ms). The value set here will define the maximum toggle time seen from the PLC before an error is triggered. The PLC needs to repeatedly write to Modbus registers 0/1 (Netinteger0) within this time period (by default the sample PLC programs will toggle a bit within these registers every 500ms – this is also customisable and is described later on)
_fScale	The user can set a scalefactor representing the number of encoder counts produced by the motor for one user unit of travel. The default value of 131072 represents a user unit of "revs" for an ESM servo motor fitted with Smartabs feedback
_nHomeType	The user can define what type of home sequence the axis performs when asked to datum (refer to the HOME keyword in the Mint help file for a full list of available home types and their associated Mint constant values)
_nHomeInput	Allows the user to specify which of the drive's local digital inputs (0,1 or 2) is to be used as the home sensor input. If a home type is used that does not require an input (e.g. _hmPOSITIVE_INDEX) then set this value to -1
_nFwdLimit / _nRevLimit	Allows the user to specify which of the drive's local digital inputs (0,1 or 2) are to be used as directional limit inputs. If the application does not require travel limits then set these constants to -1
_nLimitMode	Allows the user to specify how the drive reacts to activation of one of the limit inputs. By default the drive will crash stop and disable (refer to the LIMITMODE keyword in the Mint help file for other options and their associated Mint constant values)
_nStopInput	Allows the user to specify which of the drive's local digital inputs (0,1 or 2) is to be used as the Mint stop input (activation of the Mint stop input results in the Mint application's stop event being processed). If there is no requirement for a stop input then set this constant to -1
_nStopMode	Allows the user to specify how the drive reacts to activation of the stop input. By default the drive will decelerate to standstill at the currently defined deceleration rate (refer to the STOPMODE keyword in the Mint help file for other options and their associated Mint constant values)
_nInputLevel	Allows the user to specify whether inputs are considered to be active high or active low. The default value is specified as a binary value (input 0 is the least significant bit) setting all available input as active high. Note the use of conditional compilation to ensure the correct number of inputs for the relevant drive platform are configured
_nEncoderWrap	If the application is using a modulo axis (e.g. a turntable that requires the axis position to be wrapped in the range 0 to 360 degrees) then this constant should be set to the number of encoder counts in one full cycle of the axis. For non-modulo axes set this constant to zero
_fFolErrorFatal	Sets the magnitude of following error (difference between demand and measured position) that will result in a following error trip. This value is typically set after fine tuning when the user can recognise what may be construed as a "normal" following error during motion. The fatal value is then set somewhere above this "normal" performance
_nMasterChannel	If the application needs to be geared to (i.e. FOLLOW) a master encoder reference then this allows the user to set which encoder channel is used as the master reference
_nMasterEncMode	Allows the user to set the count direction for the master encoder channel (0 or 1 as required)
_nFollowMode	Allows the user to set the required Mint FOLLOWMODE (e.g. _fmNO_RAMP, _fmRAMP) – see the Mint Help file for the full range of available follow modes
_nMotorDirection	Allows the user to configure which way the motor rotates for a positive move command (set to 0 or 1 as required)

Mint GDI detailed description

In most applications it is unlikely that the user will need any knowledge of the Mint application (other than editing the application related data described above). However, for completeness, the Mint GDI is described in full detail in the following sections.

Data Interface

Modbus Register Offset	Function	Drive Internal Address	Drive Data Type
0/1	Command Word	NETINTEGER (0)	32 bit Integer
2/3	Command Type	NETINTEGER (1)	32 bit Integer
4/5	Value	NETFLOAT(2)	32 bit IEEE Float
6/7	Speed	NETFLOAT(3)	32 bit IEEE Float
8/9	Acceleration Rate	NETFLOAT (4)	32 bit IEEE Float
10/11	Deceleration Rate	NETFLOAT (5)	32 bit IEEE Float
12/13	Accel Jerk Rate	NETFLOAT (6)	32 bit IEEE Float
14/15	Decel Jerk Rate	NETFLOAT (7)	32 bit IEEE Float
16/17	Latch Offset	NETFLOAT(8)	32 bit IEEE Float
18/19	DO Force	NETINTEGER (9)	32 bit Integer
Modbus Register Offset	Function	Drive Internal Address	Drive Data Type
200/201	Status Word	NETINTEGER (100)	32 bit Integer
202/203	Measured Position	NETFLOAT(101)	32 bit IEEE Float
204/205	Measured Velocity	NETFLOAT(102)	32 bit IEEE Float
206/207	Following Error	NETFLOAT(103)	32 bit IEEE Float
208/209	Axis Mode	NETINTEGER (104)	32 bit Integer
210/211	RMS Current	NETFLOAT(105)	32 bit IEEE Float
212/213	Error Code	NETINTEGER (106)	32 bit Integer
214/215	DI Status	NETINTEGER (107)	32 bit Integer
216/217	DO Status	NETINTEGER (108)	32 bit Integer
218/219	Torque Actual (Nm)	NETFLOAT(109)	32 bit IEEE Float

We'll examine each of these functions in detail in the following sections…

Command Word: Modbus register offsets 0/1, NETINTEGER(0)

The PLC uses bits in the command word to perform specific operations on the drive. The Mint program automatically calls Event NETDATA0 whenever the PLC writes to the command word.

Bit	Function	State 0	State 1
0	Remote drive enable control	Disable drive (if Mint program constant _nRemoteEnable is set to _true)	Enable drive (if Mint program constant _nRemoteEnable is set to _true)
1	Enable operation*	Motion inhibited	Motion permitted
2	Latch control	Position latch disabled	Position latch enabled
3	Forward hardware limit control	Forward hardware limit input assigned (subject to setting of Mint constant _nFwdLimit)	Forward hardware limit input de-assigned
4	Reverse hardware limit control	Reverse hardware limit input assigned (subject to setting of Mint constant _nRevLimit)	Reverse hardware limit input de-assigned
5	Modulo axis	Axis has no modulo position. Absolute moves use full range of axis position	Axis position wraps to a defined modulo. Absolute moves use shortest route to wrapped position
6	Fault reset control	Unused	Rising edge to state 1 causes drive fault to be reset
7	Trigger command	Unused	Rising edge to state 1 causes the drive to action the command loaded via PLC output word 1 *
8	Watchdog	Watchdog off	Watchdog on
9	Ignore following error	Following error detection is enabled	The drive ignores following errors
10-31	Spare	Unused	Unused

Note that the trigger bit activates the command loaded in the Command Type word. All the other bits in the Command word operate continually and do not require triggering.
The Modulo axis command bit is only utilised by the MOVEA and INCA motion commands. If using a modulo axis the Mint program should be edited to set the _nEncoderWrap constant equal to the number of encoder counts in one axis cycle.
If Watchdog monitoring is enabled on the drive then the Watchdog bit should be toggled frequently enough to prevent the drive's watchdog timeout occurring (typically toggle this bit within half of the watchdog timeout value).

* Some commands don't require motion to be performed and can be issued even if the Enable Operation bit is zero (e.g. Set Position and Cancel commands)

Command Type: Modbus register offsets 2/3, NETINTEGER(1)

The PLC uses these registers to load specific commands on the drive. The Mint program example provides a number of pre-defined command types. These can easily be expanded by adding to the command enumeration list in the Mint program and including additional code in the Mint 'ActionTrigger' task.

Command Number	Function	Mint Keyword
0	Home	HOME
1	Relative move	MOVER
2	Absolute move	MOVEA
3	Run at constant speed	JOG
4	Relative incremental move	INCR
5	Absolute incremental move	INCA
6	Set position	POS
7	Follow	FOLLOW
8	Speed reference	VELREF
9	Torque reference	TORQUEREF
10	Cancel motion	CANCEL
11	Controlled Stop	STOP
12	Find End Stop	Not applicable (uses a defined sequence of commands)

The PLC should pre-load the Command Type registers with the appropriate command number (and other output words containing parameters relating to the command described later) and then trigger this command by generating a rising (0->1) edge of bit 7 of the Command word

Value: Modbus register offsets 4/5, NETFLOAT(2)

The PLC uses these registers to load information that relates to the command to be issued on the drive. The table below details how the value relates to each specific command:

Command Number	Function	Use of Value
0	Home	Homing back off ratio [1]
1	Relative move	Relative move distance
2	Absolute move	Absolute move target
3	Run at constant speed (position loop enabled)	Unused
4	Relative incremental move	Relative incremental move distance [2]
5	Absolute incremental move	Absolute incremental move target [2]
6	Set position	New position value
7	Follow	Sets gear ratio
8	Speed reference	Unused
9	Torque reference	Sets torque value (%)
10	Cancel motion	Unused
11	Controlled stop	Unused
12	Find End Stop	Torque limit (% of drive rated current) [3]

The PLC should pre-load the Value registers with the appropriate data (and the other output registers containing parameters relating to the command described both previously and later) and then trigger the command by generating a rising (0->1) edge of bit 7 of the Command word.

[1] The homing back off ratio sets a value for the Mint HOMEBACKOFF keyword (refer to the Mint help file for further details). This value is just a scalar. All other values are scaled (i.e. use units dependant on the Mint SCALEFACTOR setup on the drive).
[2] Target positions for the incremental moves can be modified according to the value stored in the 'Latch Offset' registers if the latch control bit is set in the Command word (this is detailed later in the section on Latch Offset).
[3] Torque limits in the drive are effectively in series with the CURRENTLIMIT setting so it usual to set a torque limit that results in an overall current limit reduction (e.g. if CURRENTLIMIT is set to 60% of drive rated current it would be usual to set a torque limit lower than 60% when finding the end stop)

Speed: Modbus register offsets 6/7, NETFLOAT(3)

The PLC uses these registers to load a slew speed on the drive. The table below details how the Speed value relates to each specific command:

Command Number	Function	Use of Speed
0	Home	Sets Homing speed
1	Relative move	Sets slew speed for move
2	Absolute move	Sets slew speed for move
3	Run at constant speed	Jog speed demand
4	Relative incremental move	Sets slew speed for move
5	Absolute incremental move	Sets slew speed for move
6	Set position	Unused
7	Follow	Unused
8	Speed reference	Sets speed reference (in user units/s)
9	Torque reference	Unused
10	Cancel	Unused
11	Controlled stop	Unused
12	Find End Stop	Sets slew speed for seeking end stop

Speed is a scaled quantity (i.e. the units relate to the SCALEFACTOR setup on the drive). The PLC should pre-load the Speed registers with the appropriate data (and the other output registers containing parameters relating to the command described both previously and later) and then trigger the command by generating a rising (0->1) edge of bit 7 of the Command word.

Acceleration Rate: Modbus register offsets 8/9, NETFLOAT(4)

The PLC uses these registers to load an acceleration rate on the drive. The table below details how the acceleration rate value relates to each specific command:

Command Number	Function	Use of Acceleration Rate
0	Home	Sets acceleration rate to home speed
1	Relative move	Sets acceleration rate to slew speed
2	Absolute move	Sets acceleration rate to slew speed
3	Run at constant speed	Sets acceleration rate to jog speed
4	Relative incremental move	Sets acceleration rate to slew speed
5	Absolute incremental move	Sets acceleration rate to slew speed
6	Set position	Unused
7	Follow	Unused
8	Speed reference	Sets acceleration rate to speed demand
9	Torque reference	Unused
10	Cancel	Unused
11	Controlled stop	Unused
12	Find End Stop	Sets acceleration rate to slew speed

The units for acceleration rate are user units per second^2. Note that this value must be greater than 0.001. A value outside of this range will cause the drive to enter the fault state and will report error code 3 (data out of range).

The PLC should pre-load the acceleration rate registers with the appropriate data (and the other output registers containing parameters relating to the command described both previously and later) and then trigger the command by generating a rising (0->1) edge of bit 7 of the Command word

Deceleration Rate: Modbus register offsets 10/11, NETFLOAT(5)

The PLC uses these registers to load a deceleration rate on the drive. The table below details how the deceleration rate value relates to each specific command:

Command Number	Function	Use of Deceleration rate
0	Home	Sets deceleration rate from home speed
1	Relative move	Sets deceleration rate from slew speed
2	Absolute move	Sets deceleration rate from slew speed
3	Run at constant speed	Sets deceleration rate from jog speed
4	Relative incremental move	Sets deceleration rate from slew speed
5	Absolute incremental move	Sets deceleration rate from slew speed
6	Set position	Unused
7	Follow	Unused
8	Speed reference	Sets deceleration rate from speed
9	Torque reference	Unused
10	Cancel	Unused
11	Controlled stop	Sets deceleration rate from current speed
12	Find End Stop	Sets deceleration rate from slew speed

The units for deceleration rate are user units per second^2. Note that this value must be greater than 0.001. A value outside of this range will cause the drive to enter the fault state and will report error code 3 (data out of range).

The PLC should pre-load the deceleration rate registers with the appropriate data (and the other output registers containing parameters relating to the command described both previously and later) and then trigger the command by generating a rising (0->1) edge of bit 7 of the Command word

Acceleration Jerk Rate: Modus register offsets 12/13, NETFLOAT(6)

The PLC uses these registers to load an acceleration jerk rate (S-ramp) on the drive. The table below details how the acceleration jerk rate value relates to each specific command:

Command Number	Function	Use of Acceleration Jerk Rate
0	Home	Sets acceleration jerk rate to home acceleration rate
1	Relative move	Sets acceleration jerk rate to acceleration rate
2	Absolute move	Sets acceleration jerk rate to acceleration rate
3	Run at constant speed	Sets acceleration jerk rate to acceleration rate
4	Relative incremental move	Sets acceleration jerk rate to acceleration rate
5	Absolute incremental move	Sets acceleration jerk rate to acceleration rate
6	Set position	Unused
7	Follow	Unused
8	Speed reference	Sets acceleration jerk rate to acceleration rate
9	Torque reference	Unused
10	Cancel	Unused
11	Controlled stop	Unused
12	Find End Stop	Sets acceleration jerk rate to acceleration rate

The units for acceleration jerk rate are user units per second^3. Note that this value must be greater than 0.001. A value outside of this range will cause the drive to enter the fault state and will report error code 3 (data out of range).

Setting either acceleration jerk rate or deceleration jerk rate to zero will force the drive to use a trapezoidal motion profile. If both these words contain valid non-zero values the drive will use an S-ramped motion profile.

The PLC should pre-load the acceleration jerk rate registers with the appropriate data (and the other output registers containing parameters relating to the command described both previously and later) and then trigger the command by generating a rising (0->1) edge of bit 7 of the Command word

Deceleration Jerk Rate: Modbus register offsets 14/15, NETFLOAT(7)

The PLC uses these registers to load a deceleration jerk rate (S-ramp) on the drive. The table below details how the deceleration jerk rate value relates to each specific command:

Command Number	Function	Use of Deceleration Jerk Rate
0	Home	Sets deceleration jerk rate to home deceleration rate
1	Relative move	Sets deceleration jerk rate to deceleration rate
2	Absolute move	Sets deceleration jerk rate to deceleration rate
3	Run at constant speed	Sets deceleration jerk rate to deceleration rate
4	Relative incremental move	Sets deceleration jerk rate to deceleration rate
5	Absolute incremental move	Sets deceleration jerk rate to deceleration rate
6	Set position	Unused
7	Follow	Unused
8	Speed reference	Sets deceleration jerk rate to deceleration rate
9	Torque reference	Unused
10	Cancel	Unused
11	Controlled stop	Sets deceleration jerk rate to deceleration rate
12	Find End Stop	Sets deceleration jerk rate to deceleration rate

The units for deceleration jerk rate are user units per second^3. Note that this value must be greater than 0.001. A value outside of this range will cause the drive to enter the fault state and will report error code 3 (data out of range).
Setting either acceleration jerk rate or deceleration jerk rate to zero will force the drive to use a trapezoidal motion profile. If both these words contain valid non-zero values the drive will use an S-ramped motion profile.

The PLC should pre-load the deceleration jerk rate registers with the appropriate data (and the other output registers containing parameters relating to the command described both previously and later) and then trigger the command by generating a rising (0->1) edge of bit 7 of the Command word.

Latch Offset: Modbus register offsets 16/17, NETFLOAT(8)

The PLC uses these registers to load an offset distance from a captured fast interrupt position that is subsequently used as a new target position for the drive (this function is typically used on Indexing Conveyors where the axis must always stop at a fixed distance past a reference sensor). The table below details how the Latch Offset value relates to each specific command:

Command Number	Function	Use of Latch Offset
0	Home	Unused
1	Relative move	Unused
2	Absolute move	Unused
3	Run at constant speed	Unused
4	Relative incremental move	Sets offset from latch position
5	Absolute incremental move	Sets offset from latch position
6	Set position	Unused
7	Follow	Unused
8	Speed reference	Unused
9	Torque reference	Unused
10	Cancel	Unused
11	Controlled stop	Unused
12	Find End Stop	Unused

If the PLC has set bit 2 of the Command word (to enable ‘fast’ position latching) and the drive receives a latch event during either a relative incremental move or an absolute incremental move then the drive will use the data contained in the Latch Offset word to set a new target position for the move. The move target becomes the captured (fast) position plus the defined offset distance.
The Latch Offset is a scaled quantity (i.e. the units relate to the SCALEFACTOR setup on the drive). The PLC should pre-load the Latch Offset registers with the appropriate data (and the other output registers containing parameters relating to the command described previously) and then trigger one of the incremental move commands by generating a rising (0->1) edge of bit 7 of the Command word.

DO Force: Modbus register offsets 18/19, NETFLOAT(9)

The PLC uses these registers to force digital outputs to a value (as long as there are no assigned functions within the Mint program – such as motorbrakeoutput) :

Command Number	Function	Use of Latch Offset
0	Force Digital Output 0	Unused
1	Force Digital Output 1	Unused
2	Force Digital Output 2	Unused
3	Force Digital Output 3 (e180 + e190 only with SIO-01)	Unused
4	Force Digital Output 4 (e180+ e190 only with SIO-01)	Unused
5	Force Digital Output 5 (e190 only with SIO-01)	Unused
6	Force Digital Output 6 (e190 only with SIO-01)	Unused

Status Word: Modbus register offsets 200/201, NETINTEGER(100)

The PLC uses bits in the status word to determine specific conditions on the drive. The PLC can use these bits to determine when previously issued commands have completed. The Mint program automatically populates the bits in this word.

Bit	Function	State 0	State 1
0	Enable status	Drive is disabled	Drive is enabled
1	Idle status	Drive is not IDLE*	Drive is IDLE*
2	In Position	Drive is not within IDLEPOS** value of last target position	Drive is within IDLEPOS** value if last target position
3	Motor brake status***	Motor brake is released	Motor brake is applied
4	Homed status	Drive has not completed a successful home sequence	Drive has completed a successful home sequence
5	Forward limit status	Forward limit input is inactive	Forward limit input is active
6	Reverse limit status	Reverse limit input is inactive	Reverse limit input is active
7	Fault status	No fault	Fault present on drive (refer to Error code word for additional information)
8	Stop input state	Stop input inactive	Stop input active
9	Ready to enable state	Drive is not ready to be enabled	Drive is ready to be enabled
10-11	Control mode	These two bits combined report the active control mode of the drive (Current/Speed/Position control) – refer to CONTROLMODE in the Mint help file	
12	Trigger state	Command word bit 7 is clear	Command word bit 7 is set
13	Enable operation state	Command word bit 1 is clear	Command word bit 1 is set
14	Missed Latch state	Latch (if used) has been detected	Latch (if used) has been missed
15	Fault Reset state	Command word bit 6 is clear	Command word bit 6 is set
16	Phase search status	Phase search not complete	Phase search completed
17-31	Spare		

* Refer to Mint help file topic on IDLE for details of what constitutes the idle condition (see also IDLEMODE, IDLEVEL, IDLEPOS and IDLETIME)
** Refer to Mint help file topic on IDLEPOS
*** Motor brake control is only active if Mint Workbench has been used to set the various MOTORBRAKExxxxxxx parameters associated with this function. Refer to all Mint help file topics starting with MOTORBRAKE

Measured Position: Modbus register offsets 202/203, NETFLOAT(101)

The PLC uses these registers to read the current position of the drive. This value is a scaled quantity (i.e. it uses units dependant on the Mint SCALEFACTOR setup on the drive). To allow the GDI to control modulo axes as well as non-modulo axes this register actually reflects the scaled value of ENCODER(0). This allows the apparent position of the axis to be wrapped according to the setting of ENCODERWRAP(0). Refer to the Mint help file topics on ENCODER and ENCODERWRAP for further details.

Measured Velocity: Modbus register offsets 204/205, NETFLOAT(102)

The PLC uses these registers to read the current velocity of the drive. This value is a scaled quantity (i.e. it uses units dependant on the Mint SCALEFACTOR setup on the drive). Refer to the Mint help file topic on VEL for further details.

Following Error: Modbus register offsets 206/207, NETFLOAT(103)

The PLC uses these registers to read the current positional error of the drive. This value is a scaled quantity (i.e. it uses units dependant on the Mint SCALEFACTOR setup on the drive). Refer to the Mint help file topic on FOLERROR for further details.

Axis Mode: Modbus register offsets 208/209, NETINTEGER(104)

The PLC uses these registers to read the current mode of operation on the drive. Refer to the Mint help file topic on AXISMODE for further details.

Measured RMS Current: Modbus register offsets 210/21, NETFLOAT(105)

The PLC uses these registers to read the measured RMS current being produced by the drive (in amps). Refer to the Mint help file topic on CURRENTMEAS for further details.

Error Code: Modbus register offsets 212/213, NETINTEGER(106)

The PLC uses these registers to read the current fault/error code from the drive. A zero value indicates no fault is active (also bit 7 of the Status word indicates whether a fault is present or not). Refer to Mint help file topic on ERRCODE for further details.

DI Status: Modbus register offsets 214/215, NETINTEGER(107)

The PLC uses these registers to read the status of the drive's digital inputs. This will look at the current state of the digital inputs. A bit will be set if the input is active. For edge triggered inputs, the bit will be set if an edge has been latched.

DO Status: Modbus register offsets 216/217, NETINTEGER(108)

The PLC uses these registers to read the status of the drive's digital Outputs.

Torque Actual (Nm): Modbus register offsets 218/219, NETINTEGER(109)

The PLC uses these registers to read the actual torque of the motor as calculated by the drive. This value is expressed in Nm.

Example Sequences:

The IEC 61131 PLC function blocks within the example programs included with this application note automate the sequences described below making use of the GDI extremely simple for all applications. These sequences are only included for information should the user wish to design/implement their own sequence control functions.

Enabling the drive

The Mint constant _bRemoteEnable is set to _True in the downloaded Mint program (so the PLC ultimately controls the enable state). Ensure the local interlocks (e.g. stop input, drive enable input, AC supply) are all present on the drive. If the drive is ready to be enabled the status word will indicate this via the 'Ready to Enable' bit

Set the Enable bit in the Command word

The drive should enable and bit 0 of the Status word should be set to indicate this

Issuing a Home

Ensure the drive is enabled (see above)

Load a value for the home back off ratio into the Value word

Load a value for the homing speed into the Speed word

Load a value for the homing acceleration rate into the Accel word

Load a value for the homing deceleration rate into the Decel word

If required, load values for acceleration and deceleration jerk rates

Write 0 to the Command type word to set Homing as the command type

Write a 1 to bit 1 of the Command word to allow motion to be performed

Trigger the move by writing a 1 to bit 7 of the Command word

The programmed move should take place. Use the status bits (Idle, InPos, Homed and Fault) to examine this.
Write a 0 to bit 7 of the Command word ready for the next command.

Issuing a relative move

Ensure the drive is enabled (see above)

Load a value for the move distance into the Value word

Load a value for the slew speed for the move into the Speed word

Load a value for the acceleration time into the Accel word

Load a value for the deceleration time into the Decel word

If required, load values for acceleration and deceleration jerk rates

Write 1 to the Command type word to select relative move

Write a 1 to bit 1 of the Command word to allow motion to be performed

Trigger the move by writing a 1 to bit 7 of the Command word

The programmed move should take place. Use the status bits (Idle, InPos and Fault) to examine this.
Write a 0 to bit 7 of the Command word ready for the next command.

Note: There is no need to re-issue values for all or any of the parameters that don't need to change if further moves are required – the drive will retain the current values. So for example, if the PLC just needs to issue a new move of a different distance but using the same velocity profile then it need only set a new value and generate a rising edge on the trigger command bit.

Issuing a Jog

Ensure the drive is enabled (see above) and Ensure bit 1 of the command word is set to allow motion
Load a value for the Jog Speed into the Speed word
Set acceleration/deceleration and Jerk rates as required (see previous example)
Write 3 to the Command type word to select Jog
Set bit 7 of the Command word to 1 to start the axis jogging

Jog can be stopped by either writing a value of zero, clearing the Enable Operation bit, issuing a new command, activating the local stop input on the drive (if assigned) or disabling the drive

Issuing a Follow

Ensure the drive is enabled (see above)
Ensure bit 1 of the command word is set to allow motion
Load a value for the follow ratio into the Value word
Ensure the axis is either IDLE (Bit 1 of Status word , NETINTEGER(100) is set) or already following (Axismode / NETINTEGER(104) returns a value of 128)
Write 7 to the Command type word to select Following
Set bit 7 of the Command word to 1 to start the axis following the master reference (or to issue a new follow ratio if the axis is already following)

Follow can be stopped by either writing a value of zero and triggering another follow, clearing the Enable Operation bit, issuing a new command, activating the local stop input on the drive (if assigned) or disabling the drive.

Issuing a Speed Reference

Ensure the drive is enabled (see above)
Ensure bit 1 of the command word is set to allow motion
Load a value for the speed reference (in user units/sec) into the Speed word
Load values for accel/decel/acceljerk/deceljerk as described previously under 'Issuing a Jog'
Write 8 to the Command type word to select Speed reference mode
Set bit 7 of the Command word to 1 to start the axis running at the programmed speed reference

Speed reference can be stopped by writing a speed reference of zero, clearing the Enable Operation bit, issuing a new command, activating the local stop input on the drive (if assigned) or disabling the drive. Note that until a new command is sent requiring a different mode of operation the drive will remain in speed reference mode. Should the user require the drive to return to position control mode holding its current position with full torque available the PLC should issue a relative move of zero units (Command type 1 with Value of 0).

Issuing a Torque Reference

Ensure the drive is enabled (see above)
Ensure bit 1 of the command word is set to allow motion
Load a value for the torque reference (as a percentage) into the value word
Write 9 to the Command type word to select torque reference mode
Set bit 7 of the Command word to 1 to start the axis running at the programmed torque reference

Torque reference can be stopped by clearing the Enable Operation bit, issuing a new command, activating the local stop input on the drive (if assigned) or disabling the drive. Note that until a new command is sent requiring a different mode of operation the drive will remain in torque reference mode. Should the user require the drive to return to position control mode holding its current position with full torque available the PLC should issue a relative move of zero units (Command type 1 with Value of 0)

Finding an end stop (home to torque limit)

Ensure the drive is enabled (see above)
Ensure bit 1 of the command word is set to allow motion
Load a value for the torque limit (as a percentage of the drive's rated current) into the value word
Load a value for the velocity reference used to find the end stop (in user units/sec) into the speed word (the sign of this value determines the direction of travel)
Load values for accel/decel/acceljerk/deceljerk as described previously under 'Issuing a Jog'
Write 12 to the Command type word to select find end stop mode
Set bit 7 of the Command word to 1 to start the axis running at the programmed velocity reference

Find end stop can be stopped by clearing the Enable Operation bit, issuing a new command, activating the local stop input on the drive (if assigned) or disabling the drive. Note that until a new command is sent requiring a different mode of operation the drive will remain position control mode (holding position at the determined end stop).

PLC Example Programs

An example program to control a single drive is included with this application note (written for control via Modbus TCP) which comprises three main elements:

1. Function blocks for each motion command type included in the Mint Generic Drive Interface (GDI). The function blocks are very similar in operation to the PLCopen function blocks for motion control
2. A data interface function block. This code is responsible for routing the application level data (e.g. from the function block usage) to the Modbus communication level
3. Function block to read/write Modbus data

To illustrate the use of the PLC code and the Mint GDI, two visualisations are included with each PLC example program.

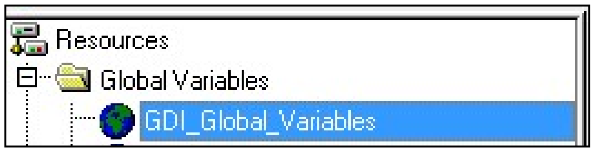
1. A visualisation to operate each of the available GDI function blocks and monitor drive status information. This visualisation is comprised of a number of smaller pre-written visualisations (also included). These can be re-used for additional axes very easily, just by assigning placeholders to the corresponding function blocks associated with the new axes
2. A visualisation to monitor the operation of the Modbus data transfer and monitor the status of Modbus communication

Creating/Defining Axes

The example PLC program is written to allow one drive to be controlled (via the GDI) over Modbus TCP. However, it is very simple to add additional axes.

For Modbus TCP the procedure to add additional axes is as follows:

1. From the “Resources” tab of the CoDeSys application double-click the “GDI_Global_Variables” icon…



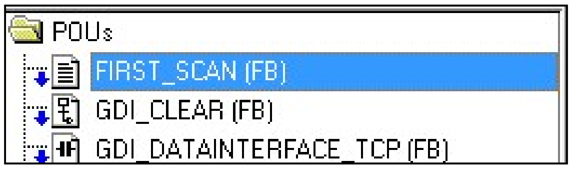
2. For every additional axis required add an additional declaration for an axis structure of type TGDIAxisRef. The example below shows two additional axes…

   (* Add axis data types as required *)
   tAxis0 : TGDIAxisRef;
   tAxis1 ; TGDIAxisRef;
   tAxis2 ; TGDIAxisRef;

3. Edit the declarations for the read and write data arrays to suit the number of axes in use. The write data uses 9 array elements and the read data uses 7 array elements, but each axis’ data starts at an index with a multiple of 10. The example below shows how these would be edited to suit 3 axes…

   (* Axis write data @ 0-8, 10-18, 20-28 : Axis read data @ 0-6, 10-16, 20-26 *)
   WriteData:ARRAY [0..28] OF DINT;
   ReadData :ARRAY [0..26] OF DINT;

4. From the “POUs” tab of the CoDeSys application double-click the “First Scan” POU…

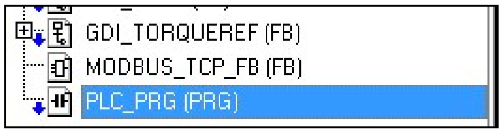


5. For every additional axis required include additional initialisation code for elements of the axis structures declared at step 2 previously. The example below shows the code that may be used for three axes of control. Note that axis numbers must be sequential (starting from zero) and that the IP address must match the actual IP addresses of the drives (as set by their configuration parameters)…

```
tAxis0.AxisName := 'Axis 0';
tAxis0.AxisNo   := 0;
tAxis0.IPAddress := '192.168.0.1';

tAxis1.AxisName := 'Axis 1';
tAxis1.AxisNo   := 1;
tAxis1.IPAddress := '192.168.0.6';

tAxis2.AxisName := 'Axis 2';
tAxis2.AxisNo   := 2;
tAxis2.IPAddress := '192.168.0.14';
```

For the example code addresses on the 192.168.0 subnet are used to suit the default used for e180/e190 drives.

6. Now double-click the PLC_PRG POU icon…



To allow the PLC to exchange control and status data with an additional drive include a call to a new instance of the GDI_DATAINTERFACE_TCP function block (passing the relevant Axis structure as a parameter to this block).

7. Add application code, as required, to the PLC program for control of the additional axis (e.g. include new instances of calls to the required GDI function blocks)

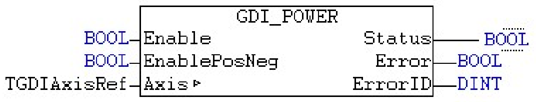
The number of axes that can be controlled by the PLC is a function of the PLC's available memory. The example programs have been written for a PM554-ETH eco-PLC which is provided with 128kB of memory for application code and 10kB of memory for variable storage.
For reference, the single axis example programs use approximately 100kB (78.13%) of the PM554's available program storage and 8.23kB (80.4%) of the available variable storage. Each additional axis uses a further 2.33kB of variable storage space.
For multi-axis applications an AC500 PLC (with greater memory capacity for program and variable storage) should be considered (e.g. PM573-ETH).

GDI Function Blocks

The following sections detail the use of the PLC GDI function blocks:

GDI_POWER

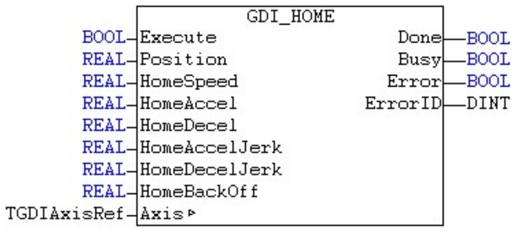
This function block is used to enable / disable an axis. The enable input enables the power stage in the drive and not the function block itself.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Enable	BOOL	Whilst True the PLC will request the axis to be enabled
EnablePosNeg	BOOL	Whist True motion in both directions is permitted. If False motion is prevented (or a stop is performed if motion is already in progress)
VAR_OUTPUT		
Status	BOOL	Indicates whether the axis is enabled (True) or not (False)
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_HOME

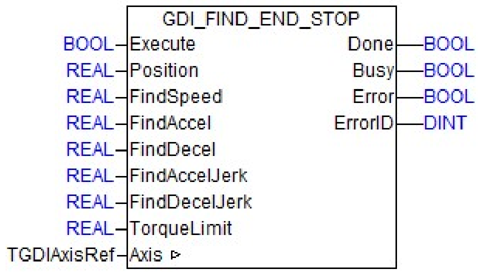
This function block is used to datum an axis to a dedicated home switch/sensor. The details of the datum sequence are dependent on the Home type set in the Mint GDI program. The Position input is used to set the axis position at the end of a successful datum sequence.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the datum sequence on a rising edge
Position	REAL	Absolute position to be set at the end of a successful datum sequence
HomeSpeed	REAL	Homing speed in user units/sec
HomeAccel	REAL	Homing accel rate in user units/sec^2
HomeDecel	REAL	Homing decel rate in user units/sec^2
HomeAccelJerk	REAL	Homing accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
HomeDecelJerk	REAL	Homing decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
HomeBackOff	REAL	Ratio of Home speed to backoff speed
VAR_OUTPUT		
Done	BOOL	Indicates that the axis has homed successfully. If the Execute input is removed during homing and the axis completes the home sequence the Done output will be set for one PLC scan. If the Execute input remains True then the Done output will also remain set (providing the home was successful)
Busy	BOOL	Set True whilst the homing sequence is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_FIND_END_STOP

This function block can be used as an alternative to GDI_HOME to datum an axis to an end of travel physical limit in the absence of a dedicated home switch/sensor. The Position input is used to set the axis position at the end of a successful datum

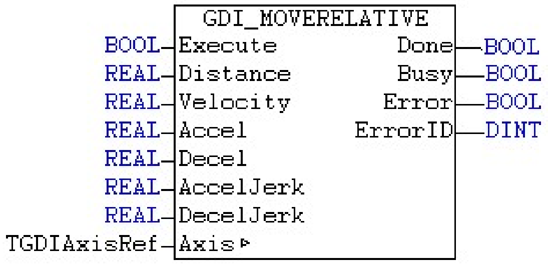


sequence.

	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the datum sequence on a rising edge
Position	REAL	Absolute position to be set at the end of a successful datum sequence
FindSpeed	REAL	Speed in user units/sec (+ve value results in +ve direction, -ve value results in –ve direction)
FindAccel	REAL	Homing accel rate in user units/sec^2
FindDecel	REAL	Homing decel rate in user units/sec^2
FindAccelJerk	REAL	Homing accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
FindDecelJerk	REAL	Homing decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
Torque Limit	REAL	Torque (current) limit expressed as a percentage of drive rated current
VAR_OUTPUT		
Done	BOOL	Indicates that the axis has found the end stop successfully. If the Execute input is removed during homing and the axis completes the find end stop sequence the Done output will be set for one PLC scan. If the Execute input remains True then the Done output will also remain set (providing the find end stop sequence was successful)
Busy	BOOL	Set True whilst the find end stop sequence is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_MOVERELATIVE

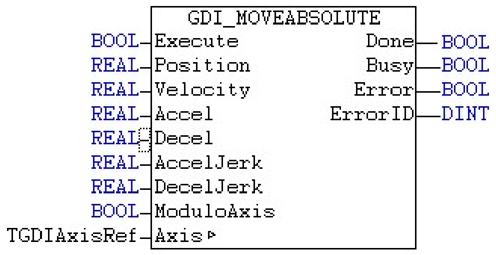
This function block is used to command a controlled motion of a specified distance relative to the set position at the time of the execution.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Distance	REAL	Relative distance for the move (in user units)
Velocity	REAL	Value for the maximum speed (not necessarily reached) in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
VAR_OUTPUT		
Done	BOOL	Indicates that the axis has reached the target position successfully. If the Execute input is removed during motion and the relative move completes the Done output will be set True for one PLC scan. If the Execute input remains True then the Done output will also remain set (providing the target position was successfully achieved)
Busy	BOOL	Set True whilst the relative move is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_MOVEABSOLUTE

This function block is used to command a controlled motion to a specified absolute position. This function can be used with Modulo axes (in which case the shortest route to the specified position will be taken).

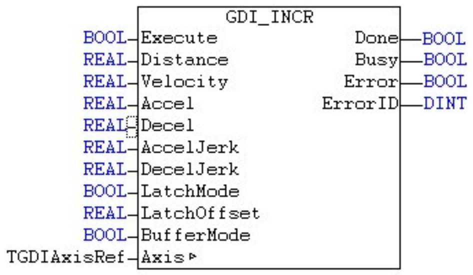


	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Position	REAL	Target position for the move (in user units)
Velocity	REAL	Value for the maximum speed (not necessarily reached) in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
ModuloAxis	BOOL	Defines whether the axis is a modulo axis (i.e. using an ENCODERWRAP to define travel within one cycle). Absolute moves when using modulo axes are always implemented via the shortest path (e.g. an absolute move to 20 degrees from 350 degrees on a 0-360 degree modulo axis will result in forward travel of 30 degrees)
VAR_OUTPUT		
Done	BOOL	Indicates that the axis has reached the target position successfully. If the Execute input is removed during motion and the absolute move completes the Done output will be set True for one PLC scan. If the Execute input remains True then the Done output will also remain set (providing the target position was successfully achieved)
Busy	BOOL	Set True whilst the absolute move is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_INCR

This function block is used to command a controlled motion of a specified distance relative to the target position at the time of the execution. The target position resulting from a call to this function block can be modified whilst motion is still in progress by any of the following methods:

a. By issuing another GDI_INCR or GDI_INCA function (providing input parameter BufferMode is True)
b. By setting the input parameter Latchmode to True and specifying a value for the input parameter LatchOffset. Mint code on the drive will then automatically modify the axis target position such that it stops the LatchOffset distance past the axis position captured by the defined fast interrupt. A bit within the Axis status word (btLatchMissed) is available to indicate failure to detect this fast interrupt (the example programs show how missing 3 latches in a row can be detected – this condition may then be used to alert the operator to a system failure for example). Using Latchmode and LatchOffset allows simple implementation of indexing conveyor applications.



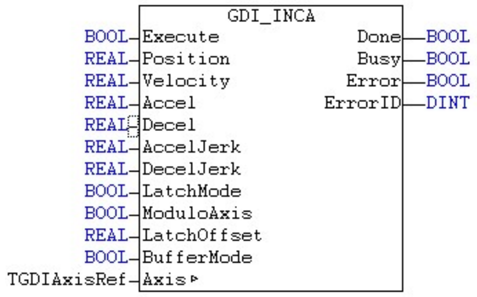
	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Distance	REAL	Relative distance for the move (in user units)
Velocity	REAL	Value for the maximum speed (not necessarily reached) in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
LatchMode	BOOL	Sets whether the axis should utilise the configured fast latch interrupt and set a new target position 'LatchOffset' user units past the captured position
LatchOffset	REAL	Defines the distance past the captured fast position (in user units) the target for GDI_INCR should be modified by (when input parameter LatchMode is set True)
BufferMode	BOOL	Defines whether the function block should set the Done output and complete as soon as the move has been loaded. Setting BufferMode True allows the application to trigger further incremental moves whilst existing moves are in progress
VAR_OUTPUT		
Done	BOOL	When BufferMode is set False this indicates that the axis has reached the target position successfully. If the Execute input is removed during motion and the relative move completes the Done output will be set True for one PLC scan. If the Execute input remains True then the Done output will also remain set (providing the target position was successfully achieved). When BufferMode is set True the Done output is set for one PLC scan to indicate successful loading of the move
Busy	BOOL	Set True whilst the function block is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_INCR is also useful if the application needs to modify SPEED/ACCEL/DECEL of a relative move already in progress. Moves loaded using GDI_MOVERELATIVE are profiled using the SPEED/ACCEL/DECEL loaded at the time and these cannot be changed once the move has started. By using GDI_INCR with the input parameter BufferMode set True then it is possible to modify the profile parameters by loading another GDI_INCR (with new SPEED/ACCEL/DECEL) with input parameter Distance set to zero.

GDI_INCA

This function block is used to command a controlled motion to a specified absolute position. This function differs from GDI_MOVEABSOLUTE in that the target position can be modified whilst motion is in progress by any of the following methods:

a. By issuing another GDI_INCR or GDI_INCA function (providing input parameter BufferMode is True)
b. By setting the input parameter Latchmode to True and specifying a value for the input parameter LatchOffset. Mint code on the drive will then automatically modify the axis target position such that it stops the LatchOffset distance past the axis position captured by the defined fast interrupt. A bit within the Axis status word (btLatchMissed) is available to indicate failure to detect this fast interrupt (the example programs show how missing 3 latches in a row can be detected – this condition may then be used to alert the operator to a system failure for example).

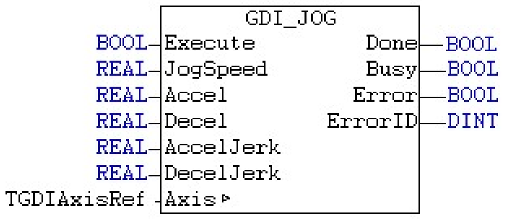


	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Position	REAL	Absolute position target for the move (in user units)
Velocity	REAL	Value for the maximum speed (not necessarily reached) in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
LatchMode	BOOL	Sets whether the axis should utilise the configured fast latch interrupt and set a new target position 'LatchOffset' user units past the captured position
ModuloAxis	BOOL	Defines whether the axis is a modulo axis (i.e. using an ENCODERWRAP to define travel within one cycle). Absolute moves when using modulo axes are always implemented via the shortest path (e.g. an absolute move to 20 degrees from 350 degrees on a 0-360 degree modulo axis will result in forward travel of 30 degrees)
LatchOffset	REAL	Defines the distance past the captured fast position (in user units) the target for GDI_INCA should be modified by (when input parameter LatchMode is set True)
BufferMode	BOOL	Defines whether the function block should set the Done output and complete as soon as the move has been loaded. Setting BufferMode True allows the application to trigger further incremental moves whilst existing moves are in progress
VAR_OUTPUT		
Done	BOOL	When BufferMode is set False this indicates that the axis has reached the target position successfully. If the Execute input is removed during motion and the relative move completes the Done output will be set True for one PLC scan. If the Execute input remains True then the Done output will also remain set (providing the target position was successfully achieved). When BufferMode is set True the Done output is set for one PLC scan to indicate successful loading of the move
Busy	BOOL	Set True whilst the function block is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_INCA is also useful if the application needs to modify SPEED/ACCEL/DECEL of an absolute move already in progress. Moves loaded using GDI_MOVEABSOLUTE are profiled using the SPEED/ACCEL/DECEL loaded at the time and these cannot be changed once the move has started. By using GDI_INCA with the input parameter BufferMode set True then it is possible to modify the profile parameters by first loading a GDI_INCA move and then loading a GDI_INCR (with new SPEED/ACCEL/DECEL) with input parameter Distance set to zero.

GDI_JOG

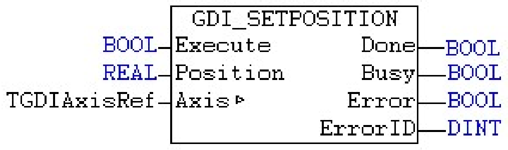
This function block is used to command a constant speed move on the axis (using the position loop controller in the drive). Motion is performed as long as the Execute input remains True.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge and maintain motion as long as the input remains True. Motion ramps to zero speed at the configured Decel rate when Execute becomes False
JogSpeed	REAL	Value for the speed the axis will reach in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
VAR_OUTPUT		
Done	BOOL	Set True as soon as the Jog command has been successfully issued and remains set until Execute becomes False or an axis error occurs
Busy	BOOL	Set True whilst the function block is in progress
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_SETPOSITION

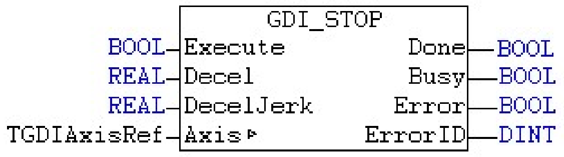
This function block is used to set the axis position (encoder and position values on the drive) to a programmed value. The axis must be idle when this function is called, otherwise the axis will return an “action not possible - motion in progress” error (Error code 10). If the axis is using an absolute encoder this will set/teach a new absolute position (GDI Mint program v2.17 onwards).



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Set the new position on a rising edge
Position	REAL	Value for the axis position to be set (in user units)
VAR_OUTPUT		
Done	BOOL	Set True as soon as the command has been issued (regardless of whether it was successful or not – use the Error output to determine whether the command was successful). Remains True until the Execute input is removed. If the Execute input is removed before the Done bit is set then the Done bit will be set for a single PLC cycle.
Busy	BOOL	Set True whilst the function block is in progress (cleared once the Done bit is set)
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_STOP

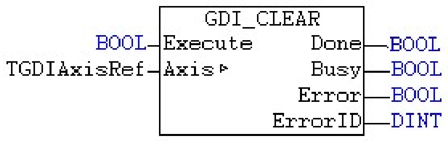
This function block is used to perform a controlled stop on the axis at the programmed deceleration rate.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the controlled stop on a rising edge
Decel	REAL	Decel rate in user units/sec^2
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
VAR_OUTPUT		
Done	BOOL	Set True when the axis becomes idle after completing the controlled stop or if an error occurs when the stop command is issued. Remains True until the Execute input is removed. If the Execute input is removed before the Done bit is set then the Done bit will be set for a single PLC cycle.
Busy	BOOL	Set True whilst the stop is in progress – cleared once the Done bit is set
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_CLEAR

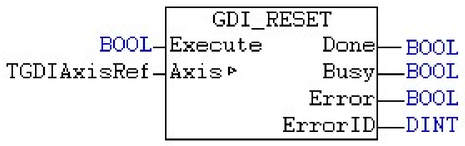
This function block is used to crash stop the axis and interrupt any motion that is in progress. The axis will remain enabled (providing GDI_POWER is requesting the enabled state and the axis is not in error).



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the crash stop on a rising edge
VAR_OUTPUT		
Done	BOOL	Set True when the axis becomes idle after completing the crash stop or if an error occurs when the crash stop command is issued. Remains True until the Execute input is removed. If the Execute input is removed before the Done bit is set then the Done bit will be set for a single PLC cycle.
Busy	BOOL	Set True whilst the stop is in progress – cleared once the Done bit is set
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_RESET

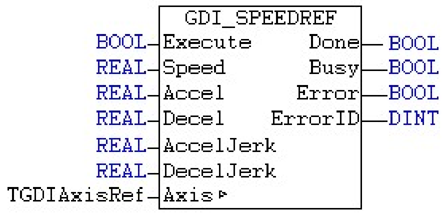
This function block is used to reset any axis error that is present.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the fault reset on a rising edge
VAR_OUTPUT		
Done	BOOL	Set True when the axis no longer has an error present. Remains True until the Execute input is removed. If the Execute input is removed before the Done bit is set then the Done bit will be set for a single PLC cycle. The Done bit will not be set if the error could not be cleared (use the Busy output to detect when the fault reset has been attempted)
Busy	BOOL	Set True whilst the function block is attempting to clear any axis error
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_SPEEDREF

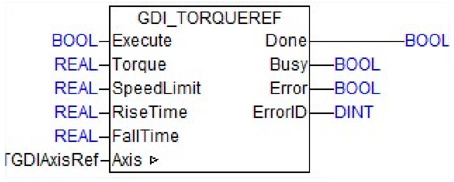
This function block is used to command a speed/velocity reference on the axis. In this mode of operation the position loop is not used on the drive (so no following error is recorded or acted upon). The axis will remain in Speed control mode until motion of another control mode type is issued. To switch from zero speed operation (in speed control mode) to holding position (in position control mode) a GDI_MOVERELATIVE could be issued, for example, with a relative move distance of zero user units.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the axis on a rising edge and maintain motion as long as the input remains True. Motion ramps to zero speed at the configured Decel rate when Execute becomes False
Speed	REAL	Value for the speed the axis will reach in user units/sec. Can be modified whilst Execute is True to change the axis speed
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
VAR_OUTPUT		
Done	BOOL	Set True as soon as the speed reference has been issued (regardless of whether it was successful or not). The Done output remains set until Execute becomes False
Busy	BOOL	Set True whilst the function block is in progress (i.e. whilst Execute is True)
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_TORQUEREF

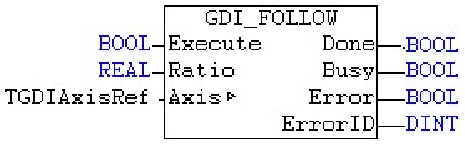
This function block is used to command a torque (current) reference on the axis. In this mode of operation, the position loop is not used on the drive (so no following error is recorded or acted upon). The axis will remain in torque control mode until motion of another control mode type is issued. To switch from zero torque operation (torque control mode) to holding position (in position control mode) a GDI_MOVERELATIVE could be issued, for example, with a relative move distance of zero user units.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the torque reference on a rising edge and maintain torque as long as the input remains True. Torque ramps to zero at the configured FallTime rate when Execute becomes False
Torque	REAL	Value for the torque reference the axis will use (in % of DRIVERATEDCURRENT – see Mint Help file). Can be modified whilst Execute is True to change the torque produced
SpeedLimit		Value of the maximum speed (in units/sec) the axis can run at given the current torque reference value.
RiseTime	REAL	Sets the time taken (in ms) for current to rise from zero to DRIVEPEAKCURRENT (see Mint Help file)
FallTime	REAL	Sets the time taken (in ms) for current to fall from DRIVEPEAKCURRENT to zero (see Mint Help file)
VAR_OUTPUT		
Done	BOOL	Set True as soon as the torque reference has been issued (regardless of whether it was successful or not). The Done output remains set until Execute becomes False
Busy	BOOL	Set True whilst the function block is in progress (i.e. whilst Execute is True)
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_FOLLOW

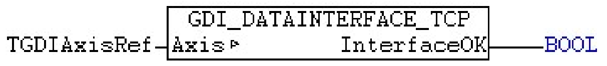
This function block is used to command the axis to start following the configured master encoder reference at the programmed follow ratio.

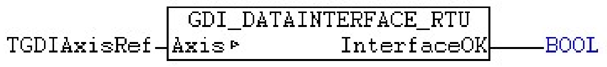


	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_INPUT		
Execute	BOOL	Start the follow on a rising edge. The axis will remain in follow mode when the Execute input becomes False (to stop the follow issue another motion command or clear motion using GDI_CLEAR)
Ratio	REAL	Value for the follow (gear) ratio between the axis and the master encoder reference (the value will affected by the scaling of the axis and the scaling of the master encoder – see the Mint Help file topic for FOLLOW). To set a new ratio whilst following it is necessary to issue a new GDI_FOLLOW command
VAR_OUTPUT		
Done	BOOL	Set True as soon as the follow has been issued (regardless of whether it was successful or not). The Done output remains set until Execute becomes False
Busy	BOOL	Set True whilst the function block is in progress (i.e. whilst Execute is True)
Error	BOOL	Set True if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_DATAINTERFACE_TCP / GDI_DATAINTERFACE_RTU

These function blocks are used to transfer command/status data between the application layer and the communication layer of the PLC programs. An instance of the relevant function block must exist for each axis in the application.

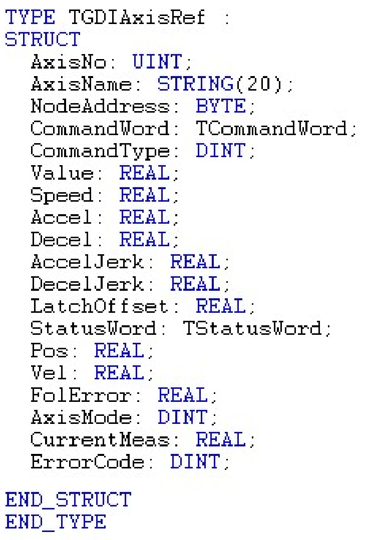




	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure (use an element of the array of axis structures when using Modbus RTU)
VAR_OUTPUT		
InterfaceOK	BOOL	True if Modbus communication is operating without error

Using the Axis Structure

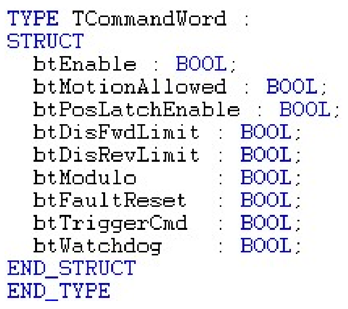
Most of the functionality of the GDI is encapsulated by the various GDI functions provided with the example PLC programs. However, in some cases the application logic may find access to the axis structure data useful (e.g. as shown in the example programs where the logic accesses the idle status and latch missed status to determine if 3 latches in a row are missed).



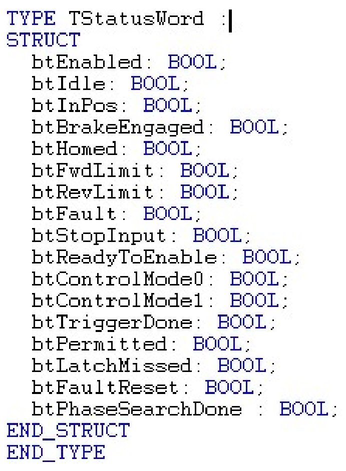
```
TYPE TGDIAxisRef :
STRUCT
  AxisNo: UINT;
  AxisName: STRING(20);
  NodeAddress: BYTE;
  CommandWord: TCommandWord;
  CommandType: DINT;
  Value: REAL;
  Speed: REAL;
  Accel: REAL;
  Decel: REAL;
  AccelJerk: REAL;
  DecelJerk: REAL;
  LatchOffset: REAL;
  StatusWord: TStatusWord;
  Pos: REAL;
  Vel: REAL;
  FolError: REAL;
  AxisMode: DINT;
  CurrentMeas: REAL;
  ErrorCode: DINT;

END_STRUCT
END_TYPE
```

The TGDIAxisRef data type deceleration is shown below:

This data structure in turn contains two further data structures (TCommandWord and TStatusWord). The declarations for these are shown below:



```
TYPE TCommandWord :
STRUCT
  btEnable : BOOL;
  btMotionAllowed : BOOL;
  btPosLatchEnable : BOOL;
  btDisFwdLimit  : BOOL;
  btDisRevLimit  : BOOL;
  btModulo       : BOOL;
  btFaultReset   : BOOL;
  btTriggerCmd   : BOOL;
  btWatchdog     : BOOL;
END_STRUCT
END_TYPE
```



```
TYPE TStatusWord :
STRUCT
  btEnabled: BOOL;
  btIdle: BOOL;
  btInPos: BOOL;
  btBrakeEngaged: BOOL;
  btHomed: BOOL;
  btFwdLimit: BOOL;
  btRevLimit: BOOL;
  btFault: BOOL;
  btStopInput: BOOL;
  btReadyToEnable: BOOL;
  btControlMode0: BOOL;
  btControlMode1: BOOL;
  btTriggerDone: BOOL;
  btPermitted: BOOL;
  btLatchMissed: BOOL;
  btFaultReset: BOOL;
  btPhaseSearchDone : BOOL;
END_STRUCT
END_TYPE
```

The PLC code can therefore access any of this data via these structures.

Example:

Reading the status of the Forward Limit Input…..

tAxis0.StatusWord.btFwdLimit

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