Motion Control Products

Application note
Generic drive interface: B&R PLC with Modbus TCP

AN00265

REV B (EN)

Ready to use PLC function blocks, combine with a pre-written Mint application for simple control of MicroFlex e190 and MotiFlex e180 drives via Modbus

Introduction

This application note provides and details an example Automation Studio project that includes library functions to allow a B&R X20 PLC to control and monitor ABB MicroFlex e190 and/or MotiFlex e180 AC servo drives via Modbus TCP. The library provides pre-written data structures and function blocks that integrate seamlessly with the Mint based GDI and allow the user to write IEC61131 based code to control a wide variety of motion on these drives. Note that MicroFlex e190 and MotiFlex e180 drives must be provided with the Mint memory card (option code +N8020).

The instructions promote consistency in all projects and greatly simplify the development of B&R PLC motion control applications where simple point to point motion is required.

This document assumes that the reader has basic knowledge of B&R PLCs, Automation Studio, Mint Workbench and the Mint GDI. It is recommended that the reader refers to application note AN00204 for details on the Mint GDI operation and configuration.

The project included with this application note provides mechanisms for a B&R X20 PLC (X20CP0410 with X20BB52 base) to:

- Issue a home command
- Issue a command to detect a physical axis end stop and use this as a datum position (drive firmware version 5863 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

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, input and output process data mappings (PDO) to NETDATA objects on the drive. Because we have used 32 bit data (UDINT data type) for the interface each value is 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.

Because a X20CP0410 processor is used, Automation studio version 4.5.2.102 or later is required to open and use the example project. The automation runtime versions used are as shown below…

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. Please refer to application note AN00204 for details.

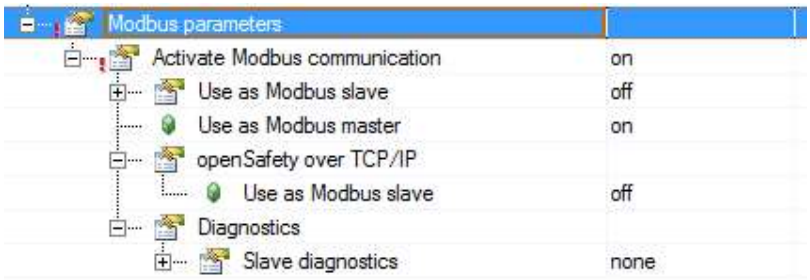
Configuring Modbus TCP on the Mint based drive

MicroFlex e190 and MotiFlex e180 drives are delivered "pre-configured" for operation of Modbus TCP via the standard Ethernet port on the front of the drive (E3). All that the user needs to do is assign a (unique) IP address to the drive via Mint Workbench to match the IP address programmed in the PLC project. In the example project provided the MicroFlex e190 is expected to be set as 192.168.0.1 (the PLC is configured with address 192.168.0.109).
When adding additional axes be sure to set unique IP addresses for each drive, remembering that these drives must all be on the same subnet as the PLC (e.g. 192.168.0.x). Use a standard Ethernet switch to connect all devices to the same network.

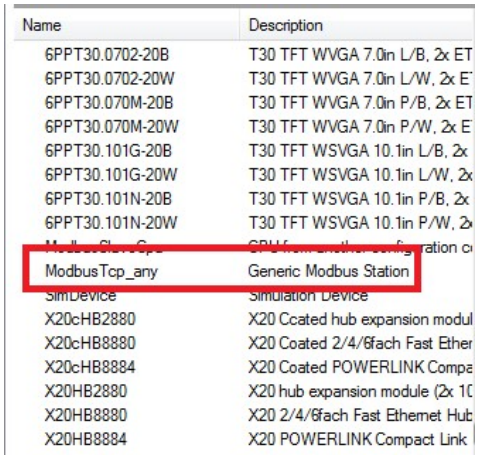
PLC configuration

The example application included with this application note shows how an X20CP0410 PLC would be configured to communicate with ABB motion drives via Modbus TCP. If starting a new application from scratch follow this process:

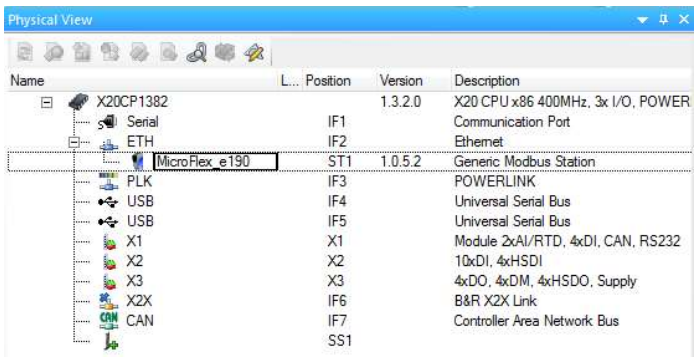
In the physical view within Automation Studio right click the Ethernet port (ETH) for the PLC and select ‘Configuration’. In the right hand pane expand the ‘Modbus parameters’ section and activate Modbus as shown below…



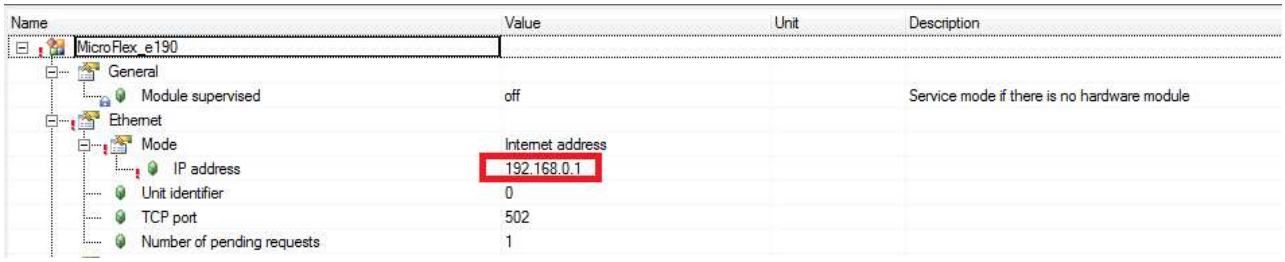
Now, with the ETH icon still highlighted in the Physical view, scroll down through the Device Catalog in the toolbox and select a ‘ModbusTcp_any’ device (and drag and drop this onto the ETH icon)….



You can rename the device that has been added if necessary (to make it clearer which drive this is)….we renamed ours to ‘MicroFlex_e190’…



Now right click the device you just added and select ‘Configuration’. The port number will already be set to 502 (because we selected a Modbus TCP device), but we will need to initially configure the IP address of the drive the PLC will be communicating with…



As our drive was configured as 192.168.0.1 we entered this IP address.

Lastly we need to add a block for the Modbus TCP read/write (Function code 23) that will be used to transfer all of the PDO data between the PLC and the drive. The starting addresses for this block and number of items (i.e. address and number of Modbus registers) must suit the Netdata locations used by the Mint GDI program on the drive. In the case of our standard GDI application the read data starts at Netdata(100) / Modbus register 200 (and there are 7 Netdata locations to read – 14 Modbus registers). The write data starts at Netdata(0) / Modbus register 0 (and there are 9 Netdata locations to write – 18 Modbus registers).

The refresh time would typically be set to half of the cycle time used for the program that transfers all Modbus data between the PLC and drives. In this example our data transfer program uses task class #1 and runs at 10ms, so we will set the Modbus refresh time to 5ms.

Our Block 1 configuration therefore ends up like this…

Block 1			
Function code	FC23: Read/Writ...		
Refresh time	5	ms	
Block send mode	cyclic		
Starting address (read)	200		
Number of items (read)	14		Set to 0 for automatic calculation
Starting address (write)	0		
Number of items (write)	18		Set to 0 for automatic calculation

Once the block is configured we can then continue to add information about each ‘Channel’ associated with this block. For each Netdata location we must add a Channel, giving this channel a name (e.g. mbStatusWord), a data type of UDINT (all the data is transferred as a 32 bit double integer initially) and a direction (Read or Write). The screenshot below illustrates some of the channel configuration…

Channel 1	
Name	mbStatusWord
Data type	UDINT
Direction	Read
Channel 2	
Name	mbMeasuredPos
Data type	UDINT
Direction	Read
Channel 3	
Name	mbMeasuredVel
Data type	UDINT
Direction	Read
Channel 4	
Name	mbFolError
Data type	UDINT
Direction	Read
Channel 5	
Name	mbAxisMode
Data type	UDINT
Direction	Read
Channel 6	
Name	mbRMSCurrent
Data type	UDINT
Direction	Read

Now right click the drive icon again and this time select ‘I/O Mapping…’ In the resulting right hand pane we need to select Process Variables for the ModuleOK channel (automatically added and indicates the operational state of the device on Modbus TCP) as well as all of the GDI PDO items we previously added via the Channel configuration…

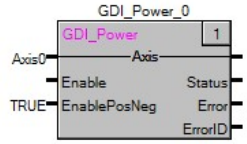
Channel Name	Process Variable	Data Type	Task Class
ModuleOk	::Axis0.NodeOK	BOOL	Automatic
mbStatusWord	::Axis0.PDOIn.pdoSTATUS_WORD	UDINT	Automatic
mbMeasuredPos	::Axis0.PDOIn.pdoMEASURED_POS	UDINT	Automatic
mbMeasuredVel	::Axis0.PDOIn.pdoMEASURED_VEL	UDINT	Automatic
mbFolError	::Axis0.PDOIn.pdoFOL_ERROR	UDINT	Automatic
mbAxisMode	::Axis0.PDOIn.pdoAXIS_MODE	UDINT	Automatic
mbRMSCurrent	::Axis0.PDOIn.pdoRMS_CURRENT	UDINT	Automatic
mbErrorCode	::Axis0.PDOIn.pdoERROR_CODE	UDINT	Automatic
mbCommandWord	::Axis0.PDOOut.pdoCONTROL_WORD	UDINT	Automatic
mbCmdType	::Axis0.PDOOut.pdoCMD_TYPE	UDINT	Automatic
mbValue	::Axis0.PDOOut.pdoVALUE	UDINT	Automatic
mbSpeed	::Axis0.PDOOut.pdoSPEED	UDINT	Automatic
mbAccel	::Axis0.PDOOut.pdoACCEL	UDINT	Automatic
mbDecel	::Axis0.PDOOut.pdoDECEL	UDINT	Automatic
mbAccelJerk	::Axis0.PDOOut.pdoACCELJERK	UDINT	Automatic
mdDecelJerk	::Axis0.PDOOut.pdoDECELJERK	UDINT	Automatic
mbOffset	::Axis0.PDOOut.pdoOFFSET	UDINT	Automatic

B&R GDI Function Blocks

The following sections detail the use of the B&R GDI function blocks:

GDI_Power

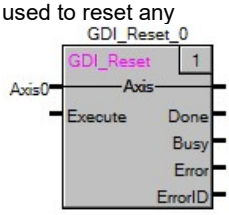
This function block is used to enable / disable an axis. The enable input enables the power stage in the drive and not the function block itself.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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 (1) or not (0)
Error	BOOL	Set to true if the axis is in error
ErrorID	DINT	Indicates the Mint error code reported by the axis

GDI_Reset

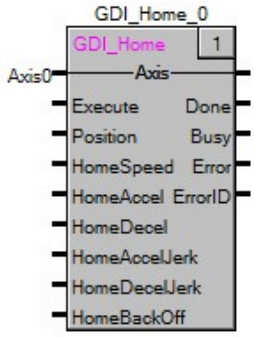
This function block is used to reset any



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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_Home

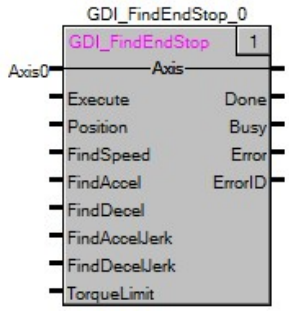
This function block is used to datum an axis. The details of the datum sequence are dependent on the Home type set in the Mint GDI program. The Position input is used to set the axis position at the end of a successful datum sequence.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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 1 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_FindEndStop

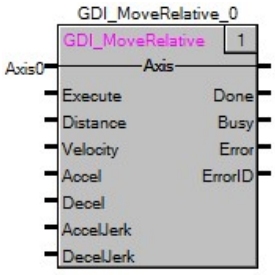
This function block is used as an alternative way to datum an axis in the absence of a home sensor. The axis will run at a commanded velocity with a programmed torque limit until this torque limit is reached and the speed of the axis is less than the programmed idle velocity. The Position input is used 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
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 (the sign of this value determines the seek direction)
FindAccel	REAL	Accel rate in user units/sec^2
FindDecel	REAL	Decel rate in user units/sec^2
FindAccelJerk	REAL	Accel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
FindDecelJerk	REAL	Decel jerk rate in user units/sec^3 (set to 0 for trapezoidal motion)
TorqueLimit	REAL	Torque limit to apply during sequence (% 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 the sequence and the axis finds the end stop the Done output will be set for one PLC scan. If the Execute input remains 1 then the Done output will also remain set (providing the sequence was successful)
Busy	BOOL	Set true whilst the find 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

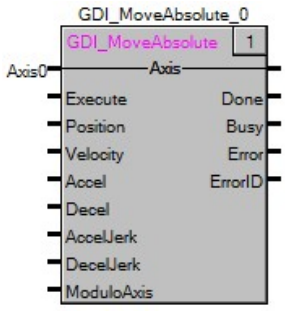
This function block is used to command a controlled motion of a specified distance relative to the start position.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Distance	REAL	Relative distance for the move (in user units)
Velocity	REAL	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 (0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (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 1 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).

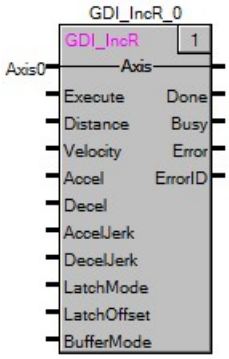


	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Position	REAL	Target position for the move (in user units)
Velocity	REAL	Maximum speed (not necessarily reached) in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (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 (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
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Distance	REAL	Relative distance for the move (in user units)
Velocity	REAL	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 (0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (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

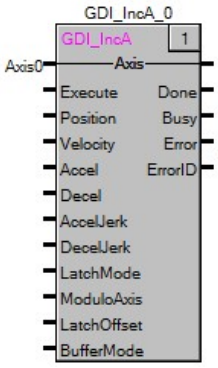
VAR_OUTPUT		
Done	BOOL	When BufferMode is set False this indicates that the axis has reached the target position successfully. If the Execute input is removed during motion and the relative move completes the Done output will be set True for one PLC scan. If the Execute input remains True then the Done 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 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 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).

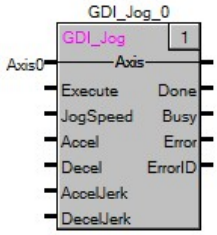


	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
VAR_INPUT		
Execute	BOOL	Start the motion on a rising edge
Position	REAL	Absolute position target for the move (in user units)
Velocity	REAL	Maximum speed (not necessarily reached) in user units/sec
Accel	REAL	Accel rate in user units/sec^2
Decel	REAL	Decel rate in user units/sec^2
AccelJerk	REAL	Accel jerk rate in user units/sec^3 (0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (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 ABB_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 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). 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

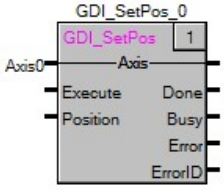
This function block is used to command a constant speed move on the axis (using the position loop controller in the drive). Motion is performed as long as the Execute input remains True.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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 (0 for trapezoidal motion)
DecelJerk	REAL	Decel jerk rate in user units/sec^3 (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_SetPos

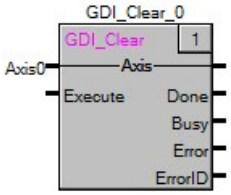
This function block is used to set the axis position (encoder and position values on the drive) to a programmed value. The axis must be idle when this function is called, otherwise the axis will return an “action not possible - motion in progress” error (Error code 10). If the axis is using an absolute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

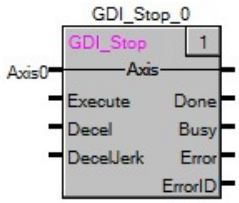
This function block is used to crash stop the axis and interrupt any motion that is in progress. The axis will remain enabled (providing GDI_Power is requesting the enabled state and the axis is not in error).



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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_Stop

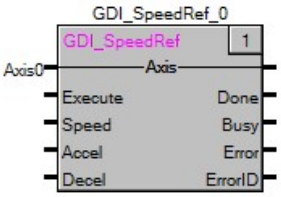
This function block is used to perform a controlled stop on the axis at the programmed deceleration rate.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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 (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_SpeedRef

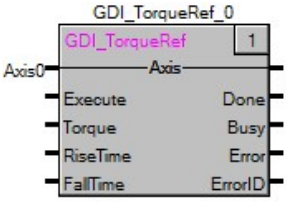
This function block is used to command a speed/velocity reference on the axis. In this mode of operation the position loop is not used on the drive (so no following error is recorded or acted upon). The axis will remain in Speed control mode (as indicated by the Statusword bits for Controlmode) until motion of another control mode type is issued (e.g. a position controlled move). 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
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
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

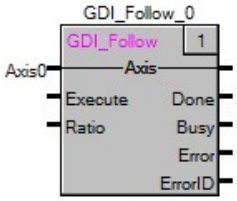
This function block is used to command a torque (current) reference on the axis. In this mode of operation the position loop is not used on the drive (so no following error is recorded or acted upon). The axis will remain in torque control mode (as indicated by the Statusword bits for Controlmode) until motion of another control mode type is issued (e.g. a position controlled move). To switch from zero torque operation (in 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
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
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

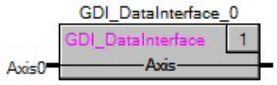
This function block is used to command the axis to start following the configured master encoder reference at the programmed follow ratio.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure
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 ABB_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 ABB_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

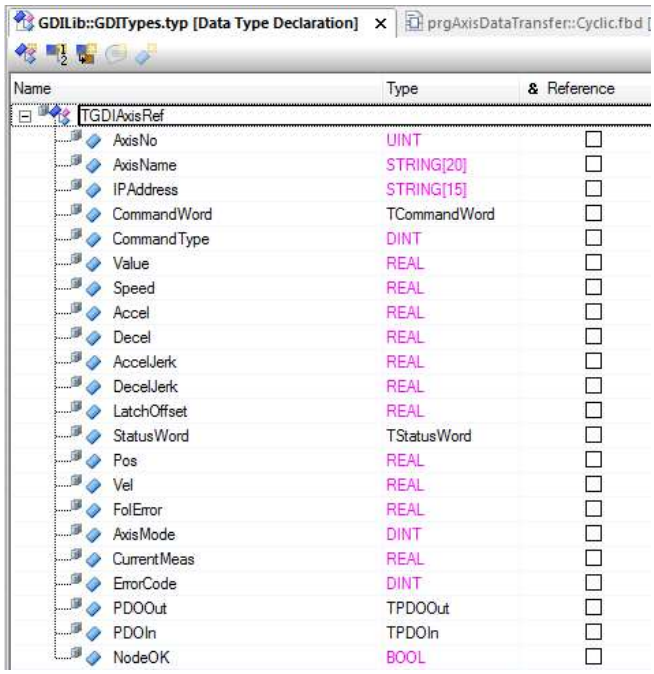
This function block is used to transfer command/status data between the PLC and the ABB motion drive. An instance of the relevant function block must exist for each axis in the application.



	Type	Description
VAR_IN_OUT		
Axis	TGDIAxisRef	Reference to the axis structure

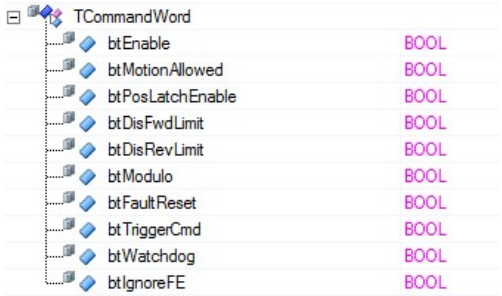
Using the Axis Structure

Most of the functionality of the GDI is encapsulated by the various GDI functions provided as library function blocks. However, in some cases the application logic may find access to the axis structure data useful. The TGDIAxisRef data type declaration is shown below:

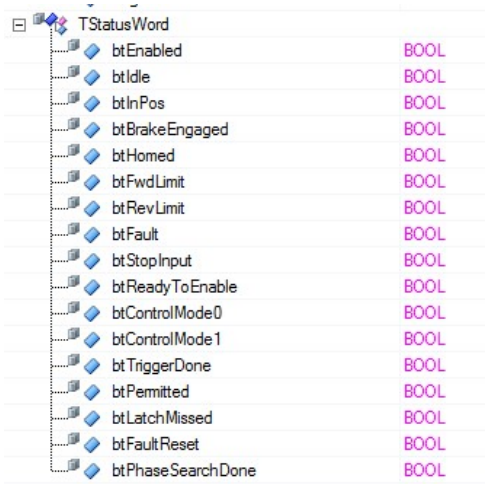


This data structure in turn contains four further data structures (TCommandWord, TStatusWord, TPDOOut and TPDOIn). The declarations for these are shown below:

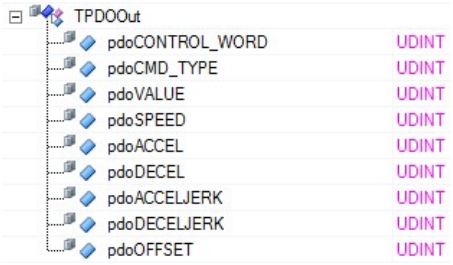
Command word



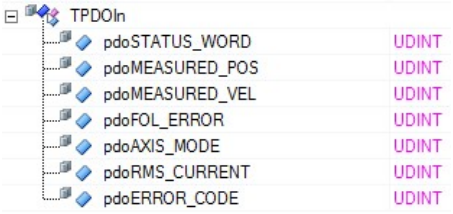
Status word



PDO Data Out (to the drive)

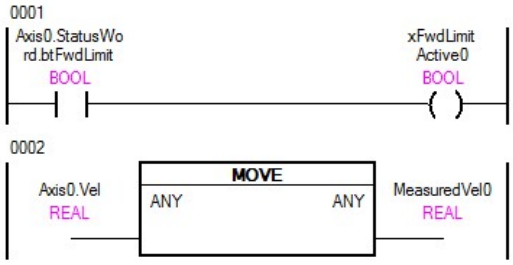


PDO Data In (from the drive)



The PLC code can therefore access any of this data via these structures, although the structures for process data are only really included to encapsulate the PDO mapping variables into the main structure to avoid the need to create unique variable names for each mapping as additional axes are added to the project and therefore wouldn't usually be accessed from the general application logic.

Example:
Two rungs accessing the axis data structure directly, one reading the status of the Forward Limit Input on the drive and the other storing the measured axis velocity…



Being able to access this data directly allows great flexibility in the PLC application code (e.g. for an indexing conveyor application the PLC application can access the latch missed status bit (btLatchMissed) and use this to drive a counter that stops motion if a certain number of latches (fast interrupts) are missed in a row).

Communication Watchdog

By default the Mint GDI is configured to use a watchdog mechanism. From receipt of the first message from the PLC the Mint program checks that communication is still active. If this is lost the axis will stop and no further moves will be possible until the error is cleared. It is possible to disable the watchdog at the drive end (see AN00204 for details), but for completeness a watchdog mechanism is included in the GDI_DataInterface function block…

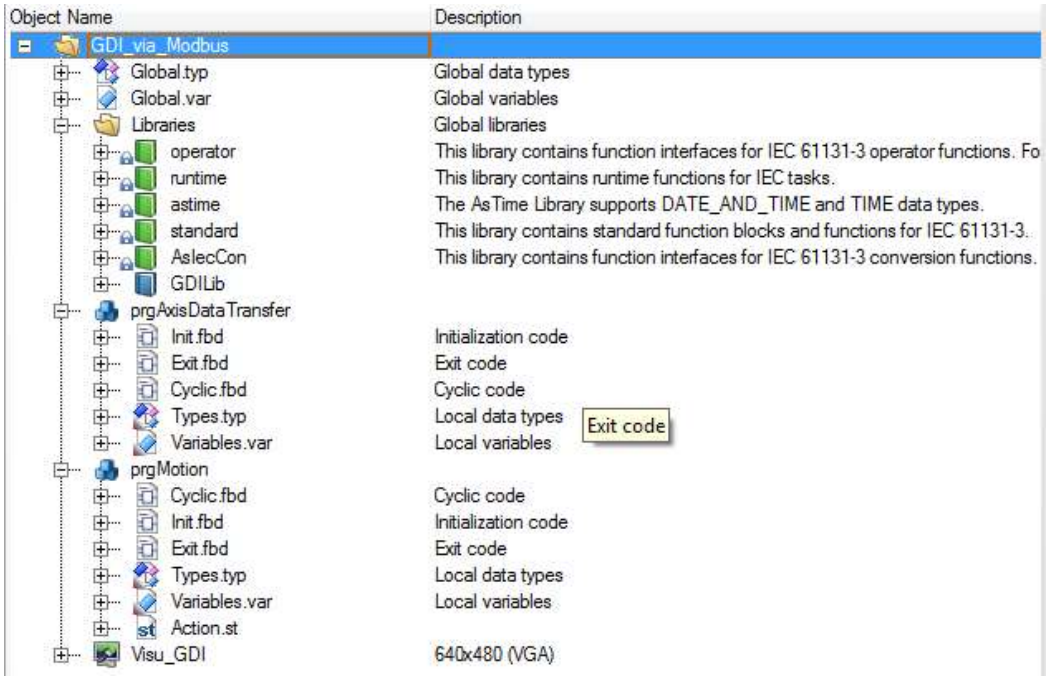
```
WatchdogBlink(Enable:=TRUE, TimeLow := _WatchdogTime / 2, TimeHigh := _WatchdogTime / 2);
WatchdogRTrig(CLK:=WatchdogBlink.Out);
IF WatchdogRTrig.Q = TRUE THEN
    iWatchdog := (iWatchdog + 1) MOD 2;
END_IF;
```

Example application

The example Automation Studio project included with this application note allows control of a single MicroFlex e190 drive (from a X20CP0410 PLC). There are two main program files (both written in Function Block Diagram / FBD)…

prgAxisDataTransfer – the cyclic.fbd element of this program calls an instance of GDI_DataInterface to transmit/receive all PDO data
prgMotion – the cyclic.fbd element of this program contains instances of every single GDI motion function block, pre-configured for use with Axis 0 (where Axis 0 is defined in Global.var as type TGDIAxisRef). The Init.fbd element of this program is used to pre-load some default values for each motion function block

These two programs are configured to run as Cyclic #1 task class (configured for 10ms cycles). This cycle time (or task class) can be adjusted to suit the application requirements and/or to suit the processor specification in use.

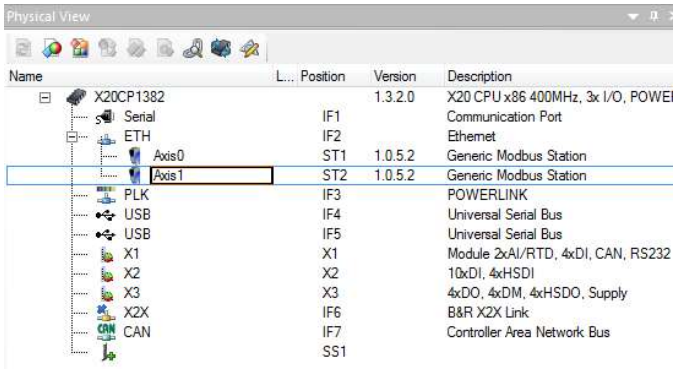


In the example application the PLC is configured with IP address of 192.168.0.109. A VNC server is configured to allow a VNC viewer (e.g. https://www.realvnc.com) to utilise the visualisation included with the project. This visualisation allows the user to test every single motion function supported by the GDI library…

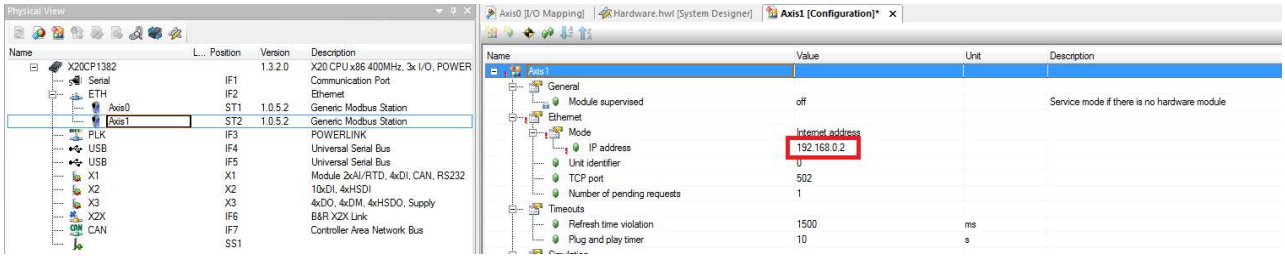
Adding additional axes

To expand the example application and add additional axes to the Ethernet (Modbus TCP) network the following simple steps should be followed:

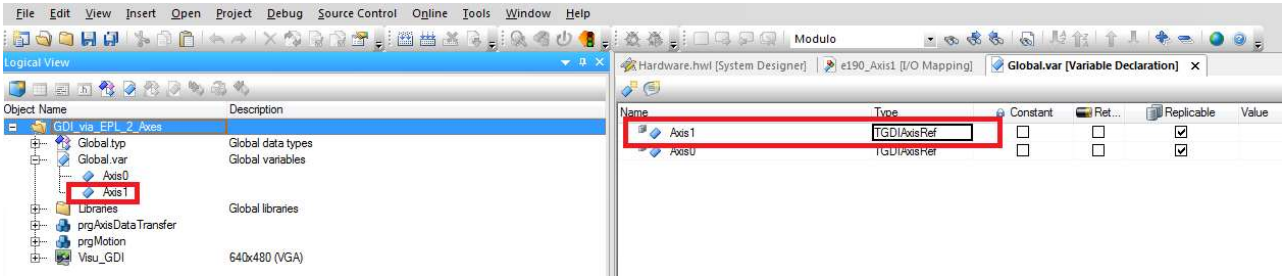
Add the new drive to the Physical View by copying and pasting the existing drive in the device tree (an additional drive should appear in the hardware (System Designer) screen as shown below)…



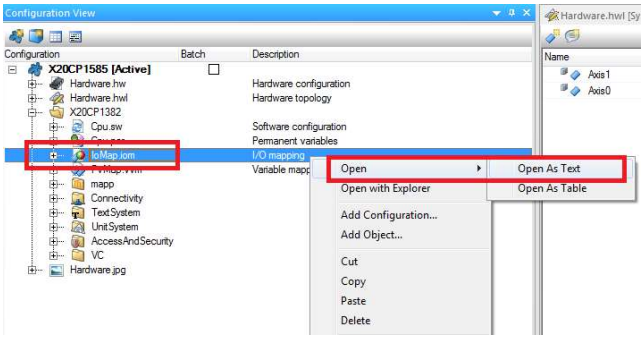
We decided to rename our drives to Axis0 and Axis1 now (as they were both MicroFlex e190 drives). Right click the new drive and select ‘Configuration’ and set the IP address to suit the additional drive…



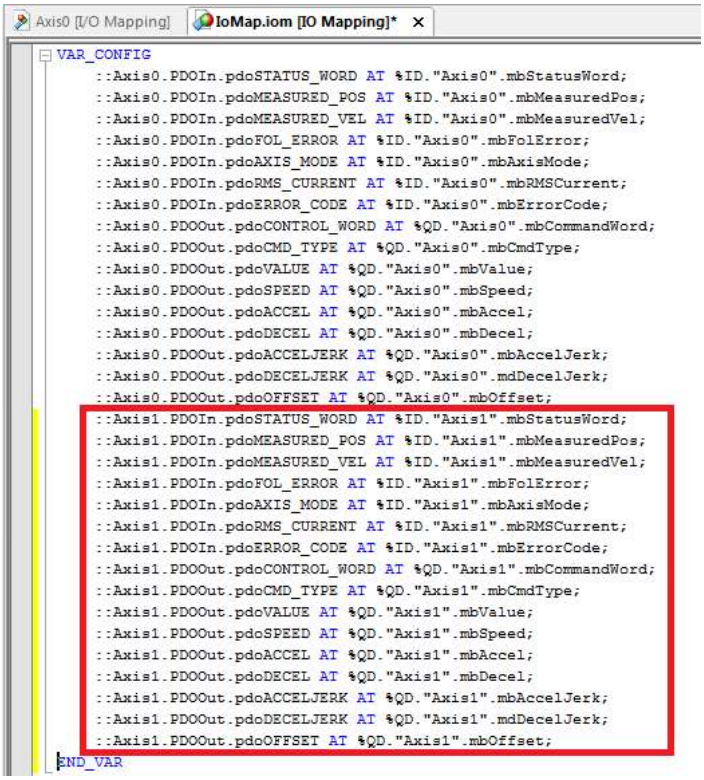
Note that by copying and pasting the drive (rather than dragging a new device from the Device Catalog) the software will automatically duplicate all of the PDO mappings that are required for the GDI interface to operate.
Now select the ‘Logical View’ in Automation Studio and add a new global variable for the new axis. This should be added as a variable of type ‘TGDIAxisRef’ as shown below…



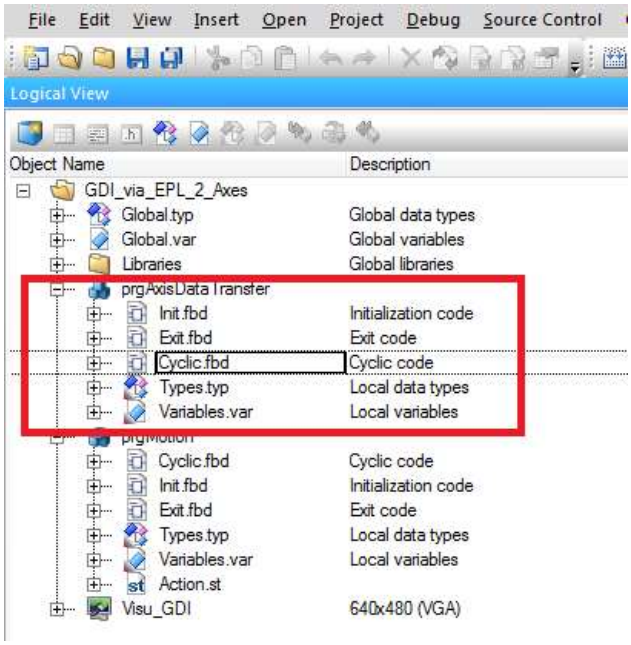
Now select the ‘Configuration View’ in Automation Studio, expand the PLC folder (X20CP0410 in our case) and right-click the IoMap.iom entry and select ‘Open>Open As Text’…



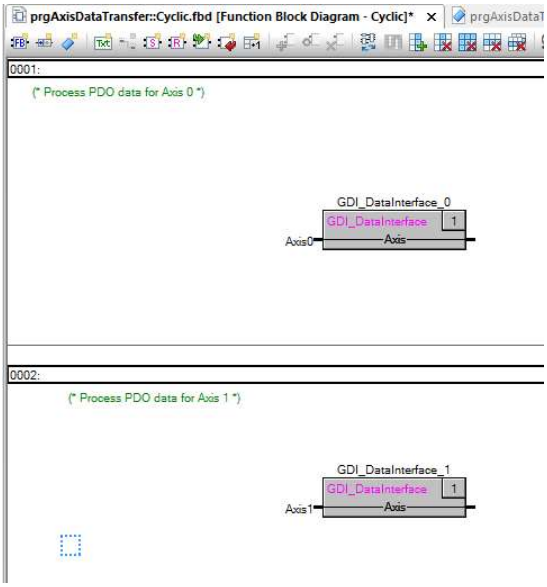
Copy all of the existing entries for the first axis (e.g. Axis0) and paste them back into the editor at the end of the file, then edit these new entries to change all references to the first axis and its associated device name so that they now refer to the new axis as shown below…

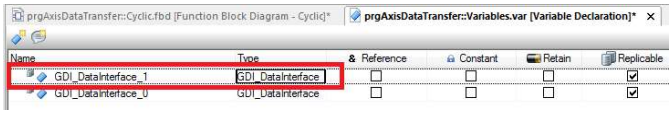


Now we need to update our program file used to transfer PDO data to/from the ABB motion drives on the Ethernet network. Switch to the 'Logical View' and double-click the prgAxisDataTransfer program's 'Cyclic.fbd' entry…



Add a new network to the Cyclic.fbd program and insert a new function block of the GDI_DataInterface type (you will find this within the GDILib section of the Libraries). You will also need to declare a new variable of this type to assign to this new function block instance…





The addition of a new axis is now complete and you are ready to start adding new application code to perform motion on this second axis, simply reference the new axis name (e.g. Axis1) in all motion function blocks.

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