Motion Control Products

Application note
AC500/ABB motion drives - EtherCAT® getting started

AN00205

Rev I (EN)

Use an AC500 PLC as an EtherCAT® master for control of MicroFlex e190 and MotiFlex e180 servo drives

Introduction

AC500 PLCs (PM585 and PM59x) can be used to perform real-time motion control of ABBs EtherCAT enabled servo drives, the MicroFlex e190 and MotiFlex e180.
This application note details how to use Automation Builder to define the hardware setup suitable for EtherCAT motion control of a single MicroFlex e190 or MotiFlex e180 and how to then write a simple AC500 PLC program to perform motion on these drives.

Pre-requisites

You will need to have the following to work through this application note:

- Mint Workbench build 5852 or later (see new.abb.com/motion for latest downloads and support information)
- A MicroFlex e190 or MotiFlex e180 drive with build 5867 or later firmware (5863 onwards is recommended as a minimum as this firmware has enhanced compliance with the CiA DS402 specification – if your drive is initially running firmware older than 5860 be sure to update to 5860 first, before updating to a later version)
- A PC or laptop running Automation Builder 1.2 or later
- An installed (and licensed) copy of the ABB PLCopen motion control library (PS552-MC-E v3.2.0 or later). Earlier versions of the library will work too but there are some small differences in the ECAT_CiA402_CONTROL_APP function block which can be worked around – contact cn-motionsupport@cn.abb.com (Asia) or motionsupport.uk@gb.abb.com (Rest of World) if necessary
- One of the following AC500 PLC processors…..PM585, PM590, PM591, PM592 or PM595 (PLC processors should be running firmware version 2.5.1 or later). The PM595 is provided with an integrated EtherCAT coupler (this should be running firmware version 4.2.32.2 or later). All other processors require a CM579-ECAT communication module (which must be running firmware version 2.6.9 or later, but ideally version 4.3.0.2 or later). Contact your local ABB PLC support team for details on how to check these requirements and update if necessary or visit http://new.abb.com/plc/programmable-logic-controllers-plcs and select the link for 'Software'. For the purposes of the text in this application note we have assumed the use of a PM591 PLC with CM579-ETHCAT coupler
- Ethernet cable to connect the PLC EtherCAT coupler to the drive

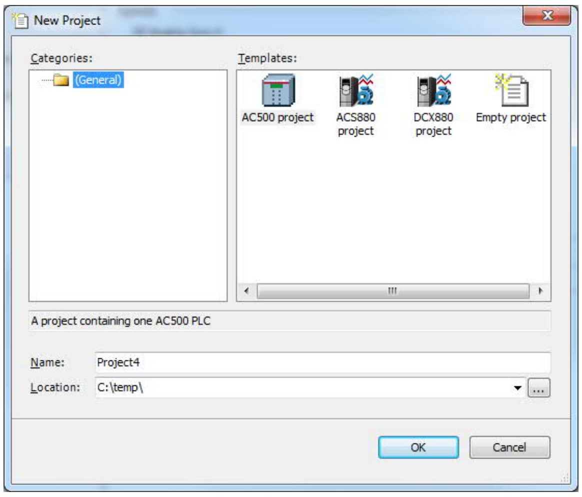
It is assumed the reader has a basic working knowledge of Automation Builder, CoDeSys and the AC500 PLC and that the connected drive has been commissioned / fine-tuned as necessary and is ready to be controlled over EtherCAT.

Drive set-up

This application note assumes that you have already commissioned the drive. That is to say you have been through the Mint Workbench commissioning wizard to define the motor and application settings and have then auto-tuned (and fine-tuned if necessary) the control loops for the drive. Details on commissioning the drive can be found in the relevant drive installation manual or you can make reference to application note AN00250. The initial operating control reference source (CONTROLREFSOURCESTARTUP) for the drive can be set to Direct or Real time Ethernet, it doesn't matter which as the PLC will always switch the drive to Real time Ethernet when EtherCAT is started. Selecting Direct as the drive's default source is preferable though as this then allows direct control of the axis via Workbench whenever the PLC is switched to the "STOP' state.

Automation Builder – Start a new PLC project

Launch Automation Builder if you have not already done so and start a new project (File>New Project…). Depending upon your Automation Builder installation there are several templates to choose from when starting a new project – select AC500 project, give your project a name and specify a location for the project (in the example below we've called the project "AN00205" to match this application note number). It's normally a good idea to create a new folder to save your project in as you may end up with more than one file associated with it.

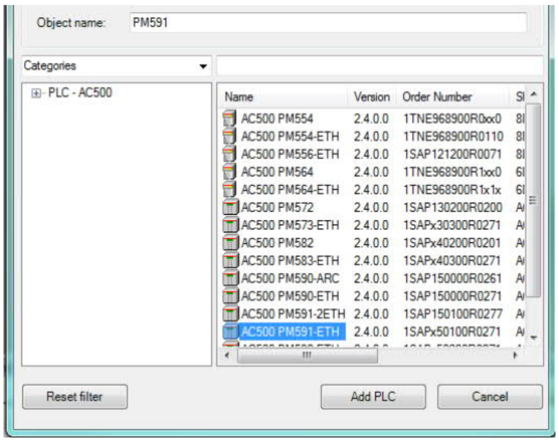


A project containing one AC500 PLC

Name: Project4

Location: C:\temp\

Click on 'OK'. Automation Builder will now ask you to select a PLC. Expand the 'PLCs' icon to select the processor type you're using. Click on 'Add PLC' to add the PLC to the project.…

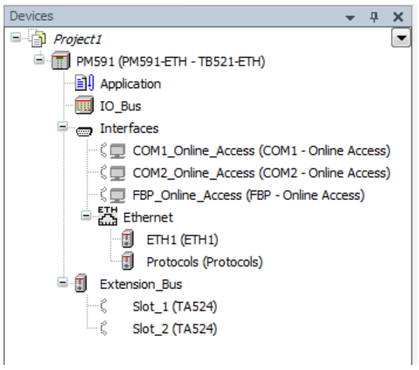


Object name: PM591

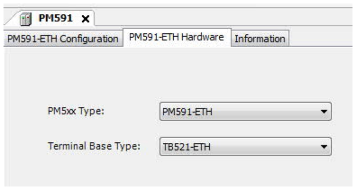
Name	Version	Order Number	SI
AC500 PM554	2.4.0.0	1TNE968900R0xx0	8I
AC500 PM554-ETH	2.4.0.0	1TNE968900R0110	8I
AC500 PM556-ETH	2.4.0.0	1SAP121200R0071	8I
AC500 PM564	2.4.0.0	1TNE968900R1xx0	6I
AC500 PM564-ETH	2.4.0.0	1TNE968900R1x1x	6I
AC500 PM572	2.4.0.0	1SAP130200R0200	A
AC500 PM573-ETH	2.4.0.0	1SAPx30300R0271	A
AC500 PM582	2.4.0.0	1SAPx40200R0201	A
AC500 PM583-ETH	2.4.0.0	1SAPx40300R0271	A
AC500 PM590-ARC	2.4.0.0	1SAP150000R0261	A
AC500 PM590-ETH	2.4.0.0	1SAP150000R0271	A
AC500 PM591-2ETH	2.4.0.0	1SAP150100R0277	A
AC500 PM591-ETH	2.4.0.0	1SAPx50100R0271	A

…once selected and it has been added to the device tree Automation Builder will automatically give the device a name based on the processor type selected. If you like you can click on the device name in the tree to overwrite this and call the device anything you like (we called ours PM591).

Automation Builder will now show the hardware structure for the PLC project….


Devices
- Project1
  - PM591 (PM591-ETH - TB521-ETH)
    - Application
    - IO_Bus
    - Interfaces
      - COM1_Online_Access (COM1 - Online Access)
      - COM2_Online_Access (COM2 - Online Access)
      - FBP_Online_Access (FBP - Online Access)
      - Ethernet
        - ETH1 (ETH1)
        - Protocols (Protocols)
    - Extension_Bus
      - Slot_1 (TA524)
      - Slot_2 (TA524)

At this point you will need select the correct terminal base. To do this double click the PM591-ETH icon which will open the PM591 field. Click on the ‘PM5xx-ETH Hardware’ tab and select the appropriate terminal base. From here you can also change the processor type if needed.


PM591

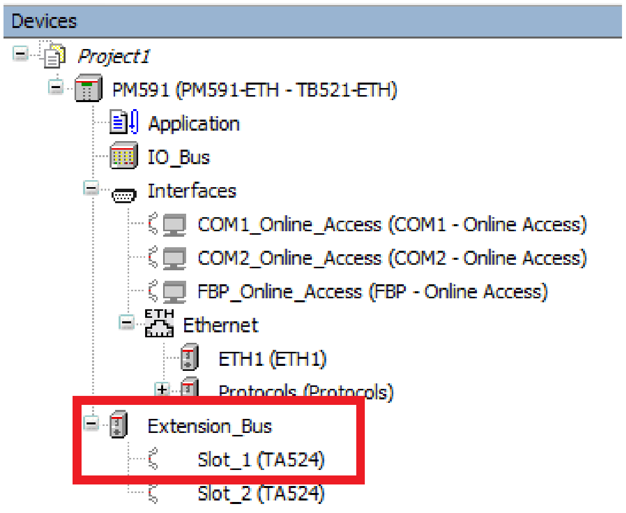
PM591-ETH Configuration | PM591-ETH Hardware | Information

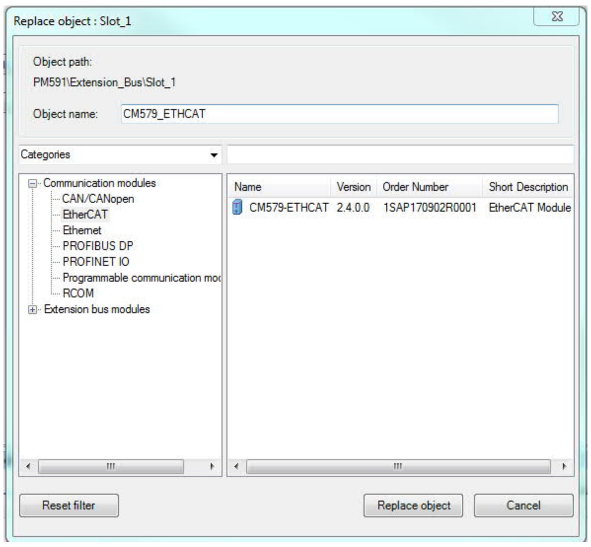
PM5xx Type: PM591-ETH

Terminal Base Type: TB521-ETH

Automation Builder – Adding EtherCAT hardware

If using a PM585, PM590, PM591 or PM592 processor, the extension bus slots to the left of the processor are numbered 1 to 4 in an ascending order (how many slots you have depends on your terminal base selection above), where 1 is closest to the processor. The CM579-ECAT must go into one of these slots. It’s most common that the CM579-ECAT module is plugged into the first slot, ‘Slot 1’. To select this right click on the ‘Slot_1 (TA542)’ icon and select ‘Add Object’. Automation Builder now presents a dialog allowing you to select which communication module is connected to this slot. Expand ‘Communication modules’ and then click ‘EtherCAT’ until you can select the CM579-ECAT module.


Devices
- Project1
  - PM591 (PM591-ETH - TB521-ETH)
    - Application
    - IO_Bus
    - Interfaces
      - COM1_Online_Access (COM1 - Online Access)
      - COM2_Online_Access (COM2 - Online Access)
      - FBP_Online_Access (FBP - Online Access)
      - Ethernet
        - ETH1 (ETH1)
        - Protocols (Protocols)
    - Extension_Bus
      - Slot_1 (TA524)
      - Slot_2 (TA524)


Replace object : Slot_1

Object path:
PM591\Extension_Bus\Slot_1

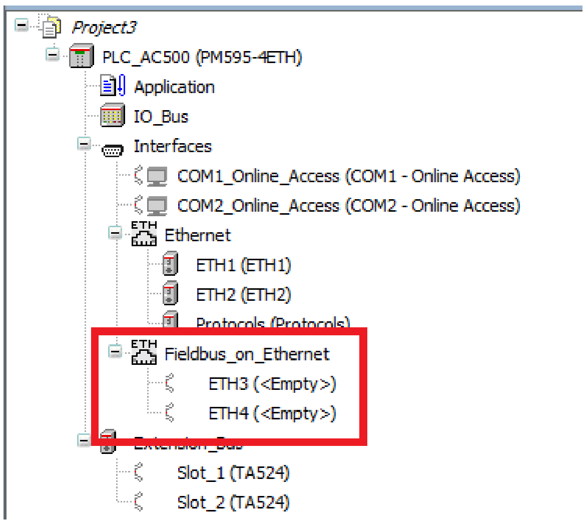
Object name: CM579_ETHCAT

Categories
- Communication modules
  - CAN/CANopen
  - EtherCAT
  - Ethernet
  - PROFIBUS DP
  - PROFINET IO
  - Programmable communication mod
  - RCOM
- Extension bus modules

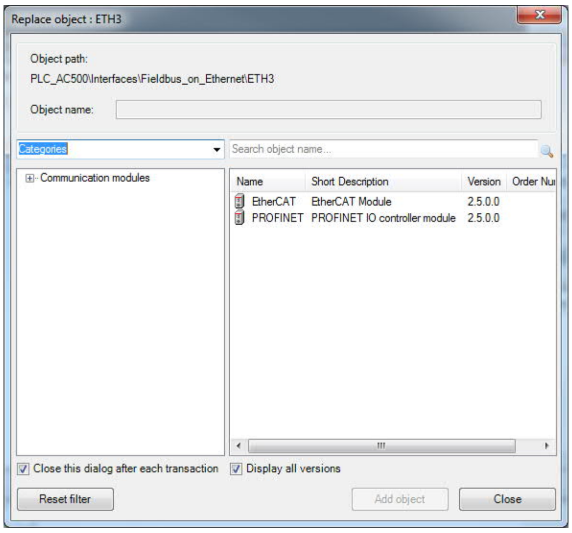
Name	Version	Order Number	Short Description
CM579-ETHCAT	2.4.0.0	1SAP170902R0001	EtherCAT Module

In the dialog box, click on ‘Replace Object’ to add this module to the hardware hierarchy.
As before, Automation Builder will name the device automatically but you can give it any name you like (we accepted the default name of “CM579-ECAT”).

If using a PM595 this has an integrated EtherCAT coupler, the CM579-ETHCAT module is not required. In the device tree right click one of the "Fieldbus_on_Ethernet" entries (e.g. ETH3) and select 'Add object'…



```
Project3
  PLC_AC500 (PM595-4ETH)
    Application
    IO_Bus
    Interfaces
      COM1_Online_Access (COM1 - Online Access)
      COM2_Online_Access (COM2 - Online Access)
      Ethernet
        ETH1 (ETH1)
        ETH2 (ETH2)
        Protocols (Protocols)
      Fieldbus_on_Ethernet
        ETH3 (<Empty>)
        ETH4 (<Empty>)
    Extension_Bus
      Slot_1 (TA524)
      Slot_2 (TA524)
```



Replace object : ETH3

Object path:
PLC_AC500\Interfaces\Fieldbus_on_Ethernet\ETH3

Object name:

Categories — Search object name...

Communication modules

Name	Short Description	Version	Order Nu
EtherCAT	EtherCAT Module	2.5.0.0	
PROFINET	PROFINET IO controller module	2.5.0.0	

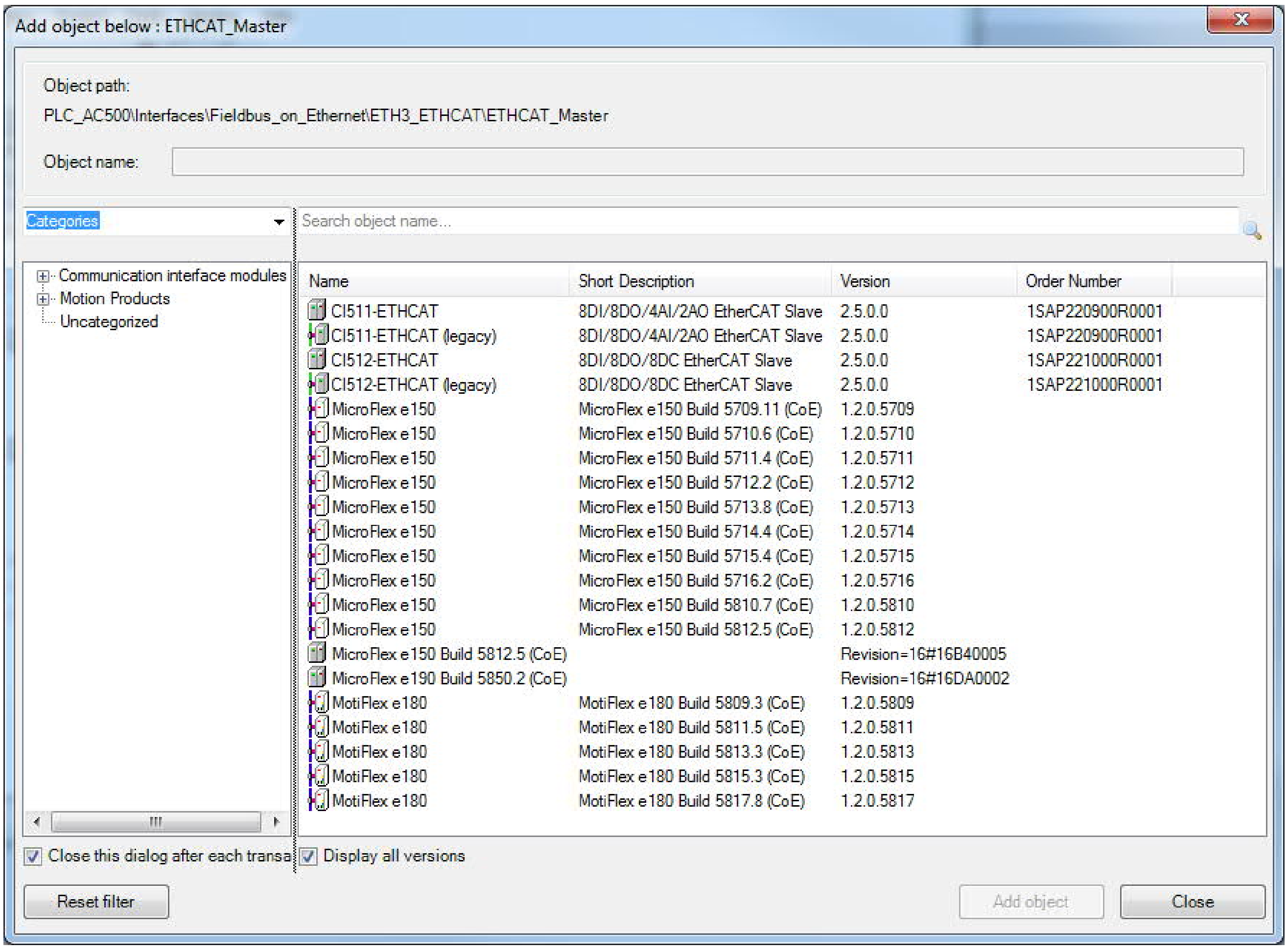
☑ Close this dialog after each transaction ☑ Display all versions

You can then select 'EtherCAT' from the available module list to add this to the selected Ethernet interface.

Now the EtherCAT master device is in the device tree we are ready to add our EtherCAT slave hardware (i.e. e190 or e180 drive).

There are two methods of adding an ABB MicroFlex e190 or MotiFlex e180 to the Automation Builder project. By selecting an object that has been installed via the Mint servo drives package or by selecting an object that has been created from an EtherCAT ESI file. We will cover both methods in the following sections of this document.

Right click the EtherCAT master device in the tree and select 'Add object'. Automation Builder will display a dialog listing all of the available EtherCAT slave devices (e.g. EtherCAT drives and EtherCAT I/O modules)….



Add object below : ETHCAT_Master

Object path:
PLC_AC500\Interfaces\Fieldbus_on_Ethernet\ETH3_ETHCAT\ETHCAT_Master

Object name:

Categories — Search object name...

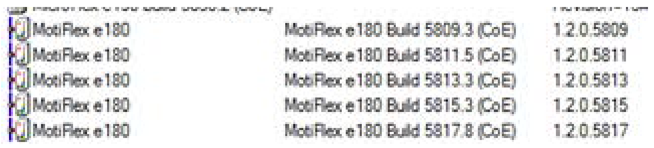
Communication interface modules
Motion Products
Uncategorized

Name	Short Description	Version	Order Number
CI511-ETHCAT	8DI/8DO/4AI/2AO EtherCAT Slave	2.5.0.0	1SAP220900R0001
CI511-ETHCAT (legacy)	8DI/8DO/4AI/2AO EtherCAT Slave	2.5.0.0	1SAP220900R0001
CI512-ETHCAT	8DI/8DO/8DC EtherCAT Slave	2.5.0.0	1SAP221000R0001
CI512-ETHCAT (legacy)	8DI/8DO/8DC EtherCAT Slave	2.5.0.0	1SAP221000R0001
MicroFlex e150	MicroFlex e150 Build 5709.11 (CoE)	1.2.0.5709	
MicroFlex e150	MicroFlex e150 Build 5710.6 (CoE)	1.2.0.5710	
MicroFlex e150	MicroFlex e150 Build 5711.4 (CoE)	1.2.0.5711	
MicroFlex e150	MicroFlex e150 Build 5712.2 (CoE)	1.2.0.5712	
MicroFlex e150	MicroFlex e150 Build 5713.8 (CoE)	1.2.0.5713	
MicroFlex e150	MicroFlex e150 Build 5714.4 (CoE)	1.2.0.5714	
MicroFlex e150	MicroFlex e150 Build 5715.4 (CoE)	1.2.0.5715	
MicroFlex e150	MicroFlex e150 Build 5716.2 (CoE)	1.2.0.5716	
MicroFlex e150	MicroFlex e150 Build 5810.7 (CoE)	1.2.0.5810	
MicroFlex e150	MicroFlex e150 Build 5812.5 (CoE)	1.2.0.5812	
MicroFlex e150 Build 5812.5 (CoE)		Revision=16#16B40005	
MicroFlex e190 Build 5850.2 (CoE)		Revision=16#16DA0002	
MotiFlex e180	MotiFlex e180 Build 5809.3 (CoE)	1.2.0.5809	
MotiFlex e180	MotiFlex e180 Build 5811.5 (CoE)	1.2.0.5811	
MotiFlex e180	MotiFlex e180 Build 5813.3 (CoE)	1.2.0.5813	
MotiFlex e180	MotiFlex e180 Build 5815.3 (CoE)	1.2.0.5815	
MotiFlex e180	MotiFlex e180 Build 5817.8 (CoE)	1.2.0.5817	

☑ Close this dialog after each transa ☑ Display all versions

Adding EtherCAT drives (Mint servo drive package method)

This is the non-preferred method as it is likely that drive firmware will be updated more frequently than the servo drive package is released, but we will detail it in this document for completeness. Drive icons with the white 'network drive style' image are entries added as part of the Mint servo drives package which can be installed as part of the Automation Builder install…



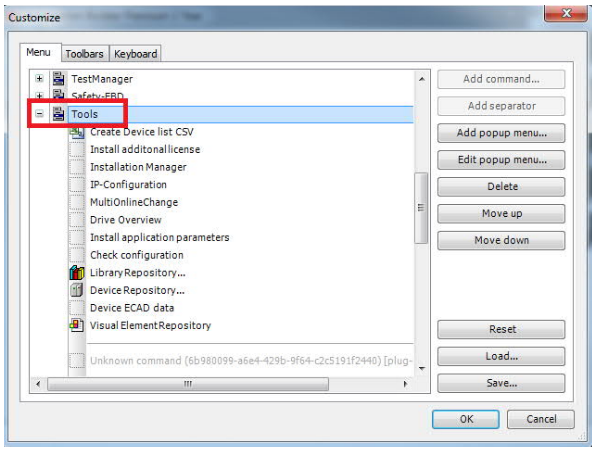
MotiFlex e180	MotiFlex e180 Build 5809.3 (CoE)	1.2.0.5809
MotiFlex e180	MotiFlex e180 Build 5811.5 (CoE)	1.2.0.5811
MotiFlex e180	MotiFlex e180 Build 5813.3 (CoE)	1.2.0.5813
MotiFlex e180	MotiFlex e180 Build 5815.3 (CoE)	1.2.0.5815
MotiFlex e180	MotiFlex e180 Build 5817.8 (CoE)	1.2.0.5817

Mint/servo drives packages contain all the Information required by Automation Builder about e190/e180 slave devices and are unique to particular firmware versions. The Mint/servo drives package allows the user to view Mint Workbench project information, include Mint files such as parameter tables within the Automation Builder project and launch Mint Workbench from Automation Builder. The latest version of the package is always available as a free download from new.abb.com/motion (on the Support page for the MicroFlex e190 or MotiFlex e180 drive, labelled as "Servo Drives package").

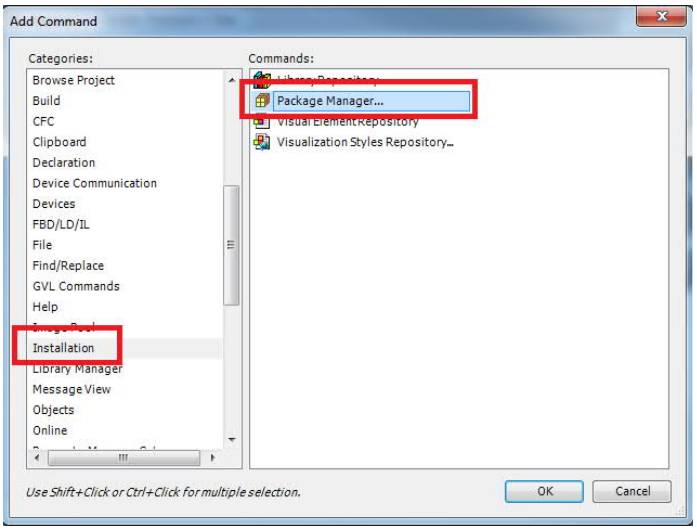
See http://www.baldormotion.com/support/SupportMe/productsupport.asp?ID=MTE180 for an example of this.

A version of the Mint/servo drives package can be installed as part of the Automation Builder install. For Automation Builder 1.2 the installed package caters for firmware versions up to 5817 (so for later versions, unless you are using a later version of Automation Builder that includes a package that contains later versions, you may need to download and install the package from the motion support site or use the file included with this application note).

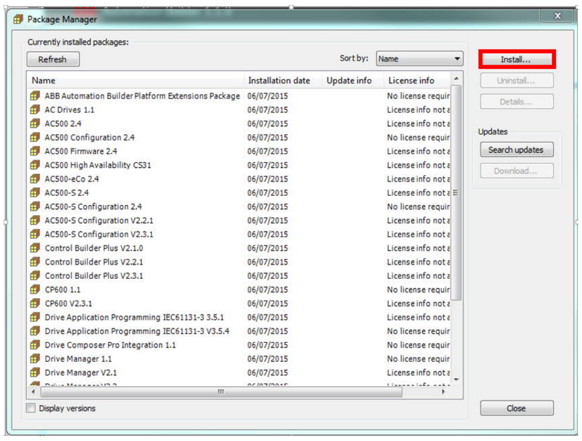
If you are going to download and install a new/later Mint servo drives package then firstly you will need access to the "Package Manager" in Automation Builder tools. To gain this access save and close the current Automation Builder project. With no project open click on the Tools > Customize menu. A warning message will pop up but you can ignore this – just click OK. The customize dialogue box will then pop up. Scroll down and expand "Tools"….



Now select any item in the list of available tools (the Package Manager will eventually appear in the Tools menu next to the item you select). Now click 'Add command…'. The Add Command dialogue box will then appear…

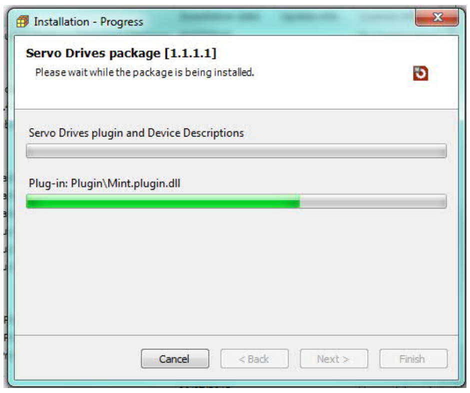


Click on ‘Installation’ in the Categories list and select Package Manager from the right hand pane and Click “OK”. Package Manager will now be visible in the tools menu ready for installing a Mint servo drives package. Assuming you have already downloaded the latest servo drives package from the motion support website you should now select Tools > Package Manager from the Automation Builder menu. The Package Manager will then open. Click Install.

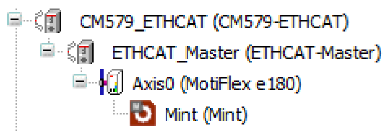


Name	Installation date	Update info	License info
ABB Automation Builder Platform Extensions Package	06/07/2015		No license requir
AC Drives 1.1	06/07/2015		License info not a
AC500 2.4	06/07/2015		License info not a
AC500 Configuration 2.4	06/07/2015		No license requir
AC500 Firmware 2.4	06/07/2015		License info not a
AC500 High Availability CS31	06/07/2015		License info not a
AC500-eCo 2.4	06/07/2015		License info not a
AC500-S 2.4	06/07/2015		License info not a
AC500-S Configuration 2.4	06/07/2015		No license requir
AC500-S Configuration V2.2.1	06/07/2015		License info not a
AC500-S Configuration V2.3.1	06/07/2015		License info not a
Control Builder Plus V2.1.0	06/07/2015		License info not a
Control Builder Plus V2.2.1	06/07/2015		License info not a
Control Builder Plus V2.3.1	06/07/2015		License info not a
CP600 1.1	06/07/2015		No license requir
CP600 V2.3.1	06/07/2015		License info not a
Drive Application Programming IEC61131-3 3.5.1	06/07/2015		License info not a
Drive Application Programming IEC61131-3 V3.5.4	06/07/2015		No license requir
Drive Composer Pro Integration 1.1	06/07/2015		No license requir
Drive Manager 1.1	06/07/2015		No license requir
Drive Manager V2.1	06/07/2015		License info not a

Navigate to your saved package and click ‘Open’. In the next dialogue box select “Typical setup” and click ‘Next’…


Installation - Progress
Servo Drives package [1.1.1.1]
Please wait while the package is being installed.
Servo Drives plugin and Device Descriptions
Plug-in: Plugin\Mint.plugin.dll
Cancel
< Back
Next >
Finish

Once installed Automation Builder will be restarted automatically for the installation to take effect. Once restarted you can now re-open your original Automation Builder project and when adding objects to the EtherCAT master device you will now see the latest list of servo drives that can be added to the device tree. Click on the appropriate Mint servo drive package object to add it to the device tree…



As with all other objects in the tree this can be given a user name (in the example above we renamed our drive as ‘Axis0’ – if asked if you want to refactor/rename all references automatically select ‘No’). Note that the Mint symbol is now attached the drive object (MotiFlex e180 in our example above). Double clicking this symbol will open the Mint tab in the right hand window from where Mint Workbench can be launched. If you’ve used this method to add a drive to the Devices tree you can skip to the ‘Automation Builder – Configuring EtherCAT settings’ section on page 8.

Adding EtherCAT drives (ESI method)

This is the preferred method for adding drives to the hardware tree as ESI files can be retrieved directly from a drive (or from the support website) – the latest ESI/device is therefore always available. Drive icons with the grey ‘PLC module style’ image are entries that have been added as a result of importing an EtherCAT ESI file...

MicroFlex e190 Build 5850.2 (CoE)
Revision=16#16DA0002

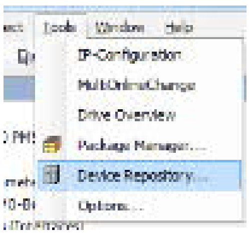
An EtherCAT Slave Information file (ESI file) is an XML format file that describes the identity and EtherCAT features of the drive and can be used by configuration tools such as Automation Builder to describe an EtherCAT slave device to the manager (the AC500 PLC). ESI files are unique to particular firmware versions of the drives. If you later update the drive to a previously unused firmware version you may need to update the Automation Builder device tree to use the new ESI based object (this could be necessary if you need to make use of new PDO mappable objects introduced with the new firmware for example).

These items allow the user to include the drive as an EtherCAT slave and configure PDO mappings and other settings to allow the drive to be controlled by the AC500 PLC, but do not allow the Mint Workbench project and associated files to be included as part of the Automation Builder project. It will always be necessary to use the ESI method if using a very new (or WIP) version of firmware for which a Mint servo drive package has not yet been created.

If Automation Builder doesn't already list an ESI file that matches the firmware version loaded on the drive you are using then ESI files can be obtained in one of three ways…

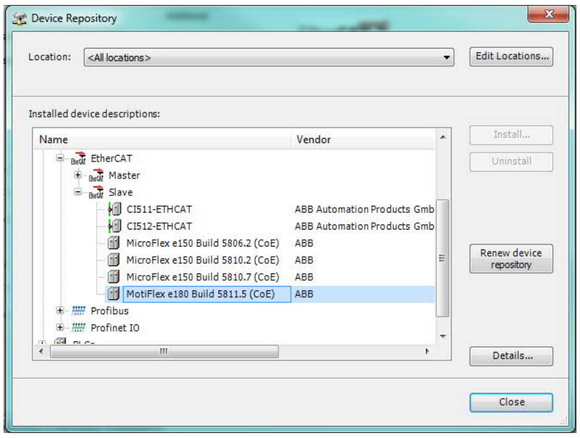
1. From the product support page on the ABB motion support website – a link to the ESI/XML file is provided
2. Using Mint Workbench connect to the drive, select the 'EtherCAT' page and on the 'Summary' tab click the 'Save As…' button to retrieve the ESI file from the drive
3. Using the Mint Sidebar application. Click on the button that connects your Internet Browser to the drive's home page. On the Home page you will find a link to the EtherCAT ESI file

However you retrieve the file, give the file a name that includes the drive's firmware version somehow (e.g. ABB MicroFlex e190 Build 5867.4 (CoE).xml) and save the file to any location you like (as long as you can remember where it is!).
Before new ESI files can be used by Automation Builder we need to install the ESI file. Select 'Device Repository…' from the Tools menu in the top menu bar



From the Device Repository dialog now expand the Fieldbuses>EtherCAT>Slave tree. Automation Builder will now display a list of EtherCAT slave modules for which it already has ESI files. If there is already an entry for the drive with firmware that matches the firmware version you're using then there is no need to install an ESI file, but by default Automation Builder does not include ABB servo drive entries. Click on the 'Install…' button if you need to add the drive.

Navigate to the drive's ESI file that you saved earlier and click on 'Open'….Automation Builder will now import the ESI file and when complete an icon representing the drive will appear in the list of slave devices…

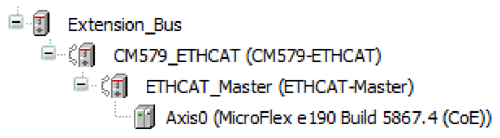


Name	Vendor
EtherCAT	
Master	
Slave	
CI511-ETHCAT	ABB Automation Products Gmb
CI512-ETHCAT	ABB Automation Products Gmb
MicroFlex e150 Build 5806.2 (CoE)	ABB
MicroFlex e150 Build 5810.2 (CoE)	ABB
MicroFlex e150 Build 5810.7 (CoE)	ABB
MotiFlex e180 Build 5811.5 (CoE)	ABB
Profibus	
Profinet IO	

Close this dialog. We can now add a drive as a slave EtherCAT device.
Right click the icon for the EtherCAT master (this was added automatically when you added the CM579-ECAT module or setup the integrated EtherCAT coupler on a PM595) and select 'Add Object…'.

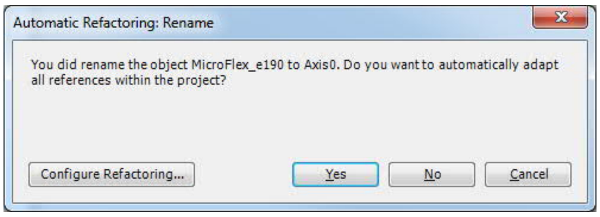
From the list of available Mint servo drive package objects and objects resulting from ESI installations select the ESI object that matches the firmware version in use on your drive and click 'Add object'.

Automation Builder will now show that the selected drive is a slave device connected to the EtherCAT master. If you wish to change the name of the drive (e.g. Axis0, Infeed, Unwind etc…) you can click on the text next to the icon in the hardware tree and enter a new name (we called ours "Axis0" in the screenshot below).



```
Extension_Bus
  CM579_ETHCAT (CM579-ETHCAT)
    ETHCAT_Master (ETHCAT-Master)
      Axis0 (MicroFlex e190 Build 5867.4 (CoE))
```

This is recommended as it will make finding PDO objects associated with this drive easier later on.
When entering a new name for the drive you may be asked if you want to automatically adapt all references within the project as shown below…



Automatic Refactoring: Rename

You did rename the object MicroFlex_e190 to Axis0. Do you want to automatically adapt all references within the project?

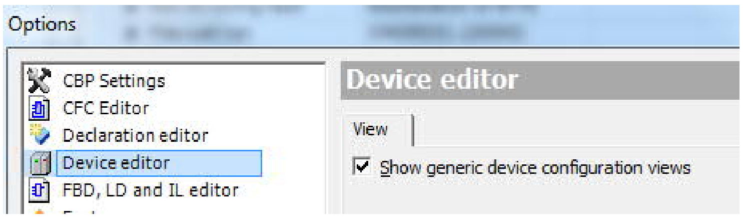
Click No (an issue with Automation Builder 1.2 results in a program crash if any other option is selected).

.

Automation Builder – Configuring EtherCAT settings

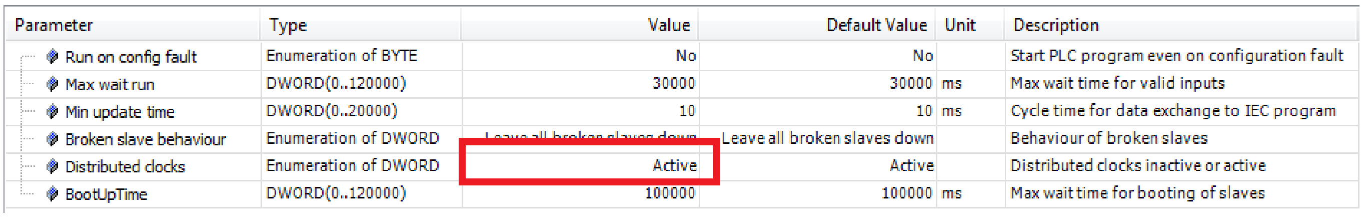
We now need to configure how our EtherCAT master and slave devices will operate (e.g. ABB EtherCAT ready servo drives support both Distributed Clock and Sync Manager type synchronisation modes. We are going to choose DC – Synchronous so we must set this and configure an EtherCAT 'bus cycle time').

First we must enable the generic device configuration view (this is needed to set cycle time less than 1 ms when using the PM595 PLC processor). From the Tools menu select Options…and then in the left hand pane select 'Device editor'. The right hand pane then shows a check box to allow you to select 'Generic device configuration view'. Make sure this is checked and click on OK…



☑ Show generic device configuration views

Now double-click the icon for the CM579_ECAT module (or the integrated EtherCAT coupler if using PM595) in the device tree. Automation Builder will display a table showing the current settings for this module. These can usually be left set at their default values, but it is important to ensure that the setting for 'Distributed clocks' is set to 'Active'…



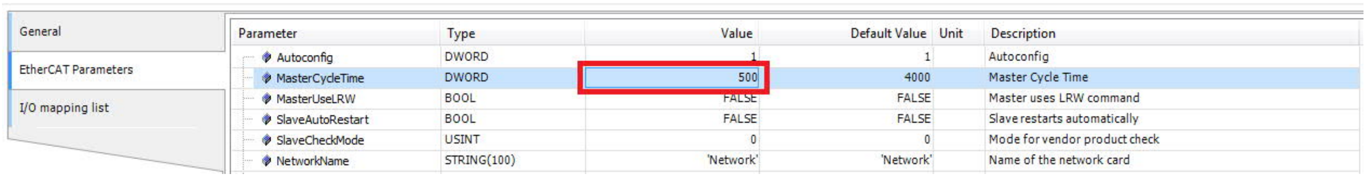
Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Max wait run	DWORD(0..120000)	30000	30000	ms	Max wait time for valid inputs
Min update time	DWORD(0..20000)	10	10	ms	Cycle time for data exchange to IEC program
Broken slave behaviour	Enumeration of DWORD	Leave all broken slaves down	Leave all broken slaves down		Behaviour of broken slaves
Distributed clocks	Enumeration of DWORD	Active	Active		Distributed clocks inactive or active
BootUpTime	DWORD(0..120000)	100000	100000	ms	Max wait time for booting of slaves

Now double click the icon below the EtherCAT coupler in the device tree labelled as '(ETHCAT-Master)'. Automation Builder now displays three tabs in the right hand pane (General, EtherCAT Parameters and I/O mapping list).

On the General tab most settings can be left as default. However there is one setting we must ensure is set correctly to suit our application…

Cycle time – this is the EtherCAT bus cycle time….this is a time in microseconds, so set this to 2000 for 2 ms cycle time for example. Note that if using a PM595 and you wish to set the minimum cycle time of 500 µs this cannot be done via this setting if using Automation Builder v1.2 or earlier. Setting 500 µs is covered later, so leave this setting unchanged if attempting to use 500 µs
Now select the 'EtherCAT Parameters' tab…

Unless you are using the PM595 and need to use a 500 µs EtherCAT cycle time there is nothing to change on this tab. If you need to set 500 µs then you should edit the value for 'MasterCycleTime' accordingly as shown below…



Parameter	Type	Value	Default Value	Unit	Description
Autoconfig	DWORD	1	1		Autoconfig
MasterCycleTime	DWORD	500	4000		Master Cycle Time
MasterUseLRW	BOOL	FALSE	FALSE		Master uses LRW command
SlaveAutoRestart	BOOL	FALSE	FALSE		Slave restarts automatically
SlaveCheckMode	USINT	0	0		Mode for vendor product check
NetworkName	STRING(100)	'Network'	'Network'		Name of the network card

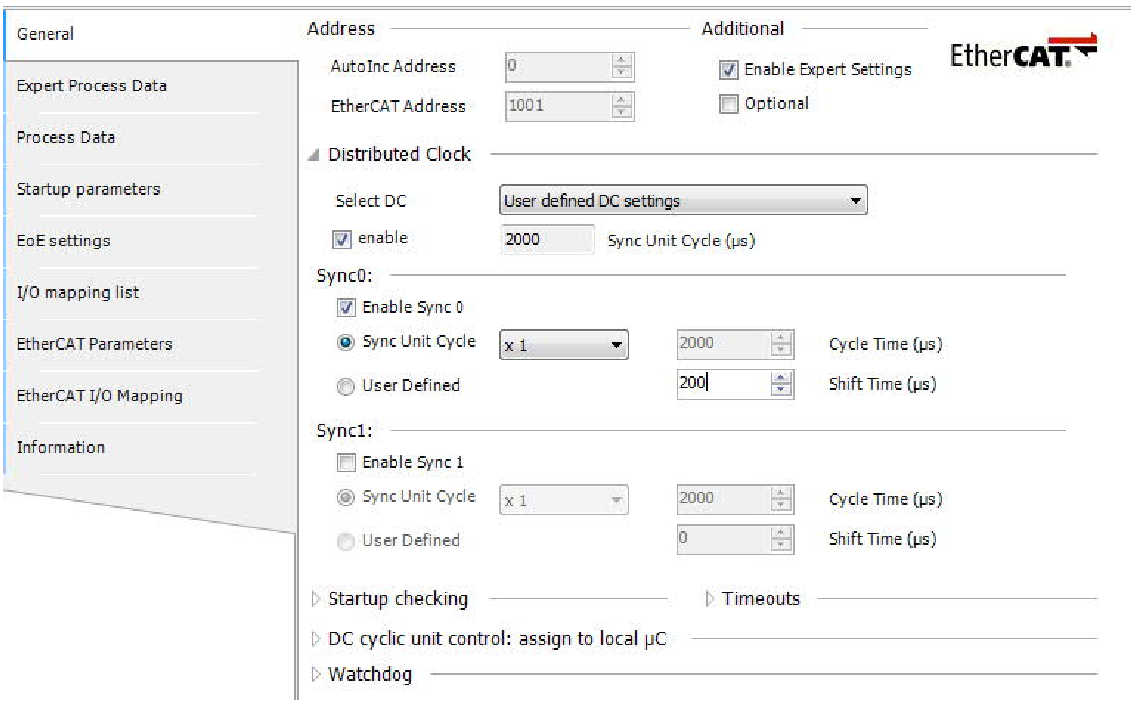
For EtherCAT cycle times >= 1 ms the value here should match the value set earlier on the General tab.

IMPORTANT : The motion drives only support binary multiples of this cycle time (e.g. 500ms, 1ms, 2ms, 4ms, 8ms etc…)

Now double-click the icon for the e190 or e180 drive (ours was renamed to Axis0 earlier). Automation Builder will then display seven tabs on the right hand side of the screen (General, Process data, Startup parameters, I/O mapping list, EtherCAT Parameters, EtherCAT I/O Mapping and Information).

Select the General tab. This tab will show the EtherCAT address of the drive (1001 by default)….this cannot be changed. As further devices are added to the network their address will automatically be set (1002, 1003 etc…). It is important to note that the physical position of the drive on the network is important. Once the network is configured do not swap the order of any EtherCAT cabling as this will effectively change the addressing of the slave devices (and therefore their configuration may not match your expected settings).

The main settings on the General tab we must set/check are the Distributed Clock scheme (set this to 'DC-Synchronous') and the 'Shift Time (us)' value. The Shift Time should typically be set to 10% of the EtherCAT cycle time (e.g. for a 2 ms cycle time set the Shift Time to 200 µs). However, by default this setting is greyed-out, so select the 'Enable Expert Settings' in order to be able to edit this value (this will also add two new tabs, Expert Process Data and EoE settings)….



Address
AutoInc Address: 0
EtherCAT Address: 1001

Additional
[✓] Enable Expert Settings
[ ] Optional

Distributed Clock
Select DC: User defined DC settings
[✓] enable 2000 Sync Unit Cycle (µs)

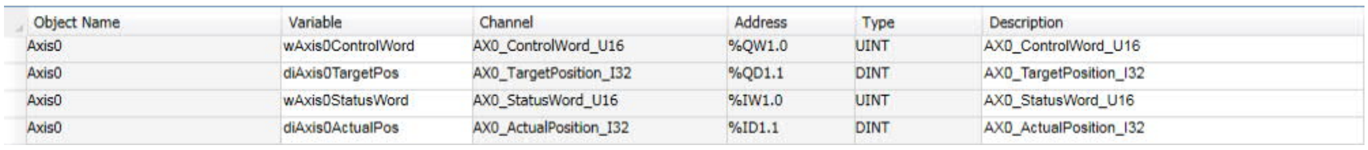
Sync0:
[✓] Enable Sync 0
(●) Sync Unit Cycle x 1 — 2000 Cycle Time (µs)
( ) User Defined — 200 Shift Time (µs)

Sync1:
[ ] Enable Sync 1
(●) Sync Unit Cycle x 1 — 2000 Cycle Time (µs)
( ) User Defined — 0 Shift Time (µs)

Now click on the Expert Process Data tab. The ESI file for the e190 and e180 drives details that for control of the drive (using Cyclic Synchronous Position) only four Process Data Objects (PDOs) are required - control word and target position from the PLC to the drive, status word and actual position from the drive to the PLC. These PDO mappings are automatically included in the project. For this simple 'Getting Started' guide there is no need to change these or add additional PDO mappings. Additional PDO mappings would be required, for example, if the application required fast position latching (i.e. Touchprobe objects). In this case the Expert Process Data page would be used to add the additional input and output objects. This is beyond the scope of this application note and is covered by other application notes (see AN00220 for example which includes a description for adding Touchprobe objects).

Now select the ‘Startup parameters’ tab. Again, there are settings here that we don’t need to change but we will detail these for reference.
‘Give EtherCAT control’ is set to 1 and this ensures that when the PLC program is started (and EtherCAT configuration data is sent to the drive) that the drive’s control reference source is automatically switched to ‘Real-time Ethernet’.
ModesOfOperation is set to 8 (for Cyclic Synchronous Position) – this is the mode needed for the PLC to profile motion (sending new target positions every bus cycle). Cyclic Synchronous Velocity (CSV/9) and Cyclic Synchronous Torque (CST/10) modes are also possible but we aren’t going to detail these in this application note.

Now select the ‘I/O mapping list’ tab. This tab lets you define variable names that the PLC program will use to address the input and output PDOs for the drive. We need to enter variable names for the four PDOs we saw were mapped earlier. Double-click in the grid under the Variable column and enter meaningful variable names to use later. The screenshot below shows how we decided to name our variables…



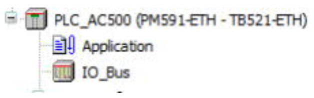
Object Name	Variable	Channel	Address	Type	Description
Axis0	wAxis0ControlWord	AX0_ControlWord_U16	%QW1.0	UINT	AX0_ControlWord_U16
Axis0	diAxis0TargetPos	AX0_TargetPosition_I32	%QD1.1	DINT	AX0_TargetPosition_I32
Axis0	wAxis0StatusWord	AX0_StatusWord_U16	%IW1.0	UINT	AX0_StatusWord_U16
Axis0	diAxis0ActualPos	AX0_ActualPosition_I32	%ID1.1	DINT	AX0_ActualPosition_I32

…we used prefixes according to the data type of the objects (as defined by AN00244) followed by “Axis0”, as that was the name of our slave drive device, and then included text to describe the function of the PDO.

When we launch CoDeSys later (the PLC programming environment) Automation Builder will automatically create a global variables module that includes these variable names. We are now ready to build the PLC configuration and launch the CoDeSys programming environment but it’s worth saving the project first – click on the floppy disk icon at the top of the screen (or press CTRL+S) to save the project.

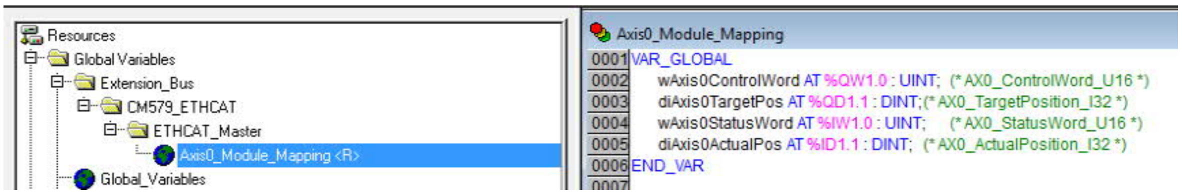
Automation Builder – Build the configuration

In the Automation Builder device tree view there’s an icon representing the PLC program associated with the project. By default this icon is named ‘Application’. If you select this icon and then single click it again it can be renamed if required (we’ll just leave it named as Application for this example).



Double-click the program icon and Automation builder builds the hardware configuration based on all the devices we’ve added. If there are errors with the hardware configuration an error message will be displayed at the bottom of the screen and you will need to fix this before proceeding (refer to the Automation Builder Help file for further information if needed). If there are no errors then once Automation Builder has built the configuration it launches the CoDeSys programming environment (remember that it is a pre-requisite that the ABB PS552-MC-E v3.1.0 PLCopen motion control libraries are already installed – contact your local ABB Sales office if you do not have these).

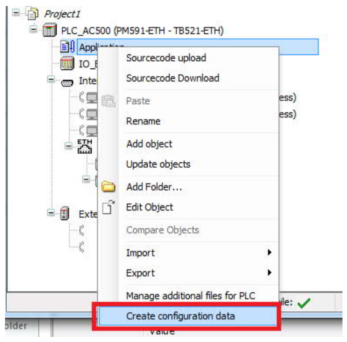
To check the variable mappings we added for the drive have been included in the PLC program automatically, select the ‘Resources’ tab in the CoDeSys project navigator and then expand the Global variables tree (the structure of which matches the hardware tree in Automation Builder) until you find the entry for ‘xxxx_Module_Mapping’ (where xxxx is the name you gave your device earlier – e.g. Axis0_Module_Mapping). Double-click this icon and the right hand pane will show the declared variables and their associated addresses as shown below:



```
VAR_GLOBAL
    wAxis0ControlWord AT %QW1.0 : UINT;  (* AX0_ControlWord_U16 *)
    diAxis0TargetPos AT %QD1.1 : DINT;(* AX0_TargetPosition_I32 *)
    wAxis0StatusWord AT %IW1.0 : UINT;    (* AX0_StatusWord_U16 *)
    diAxis0ActualPos AT %ID1.1 : DINT;  (* AX0_ActualPosition_I32 *)
END_VAR
```

If the mapped variables are not present you should close CoDeSys, return to Automation Builder and check your servo drive device has been added to the device tree correctly and that you’ve added the variable mappings as described earlier.

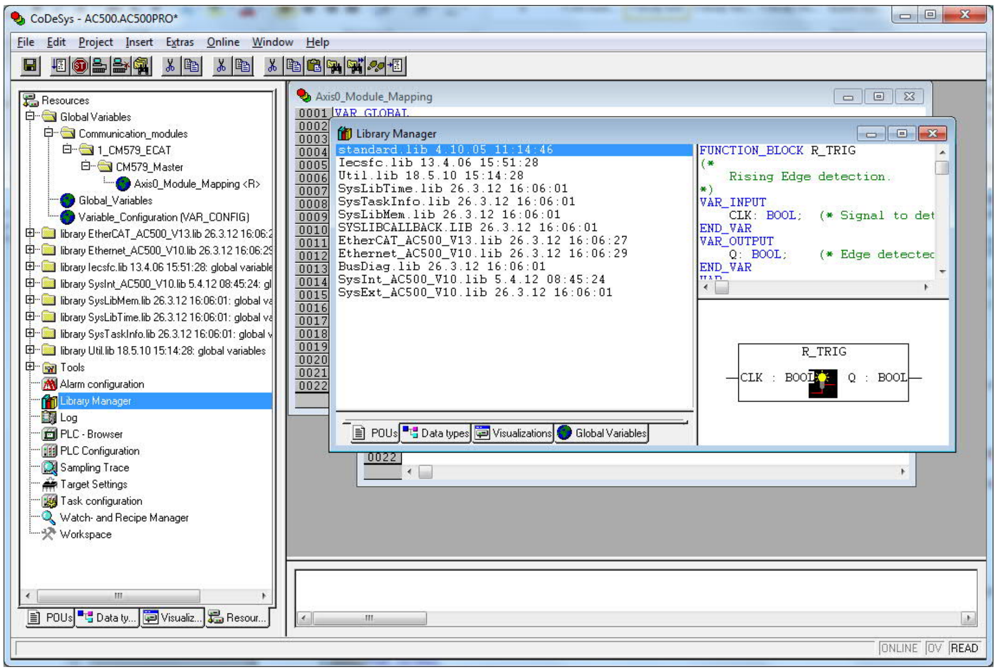
If the data is correct but hasn’t been pulled through to CoDeSys then please recreate the configuration data by opening Automation Builder then right clicking on the ‘Application’ icon and selecting ‘Create configuration data’…



At this point everything has been done to allow the EtherCAT network to become operational (providing your EtherCAT cabling is connected to the correct Ethernet ports on the PLC and drive – check the user manuals if unsure about these connections). If you like you can check the operation of the network by logging into the PLC from CoDeSys (i.e. downloading the PLC program) and running the code (please refer to the instructions on page 23 – Testing the PLC program – if you aren't sure how to go online to the PLC and download the program). If the project fails to build errors will appear in the CoDeSys build window – refer to the PLC documentation / Help system for further information on how to resolve these if necessary. Once the program is downloaded and running the green "Net Run" LED on the drive will flash a few times and should eventually turn solid green indicating that EtherCAT is operational. If this doesn't happen retrace the previously described steps. If you cannot identify the error contact your local ABB office for further support. If the LED becomes solid green then you can continue with the rest of this application note.

Using CoDeSys to write a simple program

The basic hardware setup and associated variable creation is complete, we're now ready to start writing some PLCopen code that we can use to enable our drive and perform some simple motion.
The first thing we need to do is include the PS552-MC-E library files in our PLC project. Click on the 'Resources' tab of the CoDeSys project navigator and then double-click the 'Library Manager' icon. The right-hand pane will now show which libraries are already included in the project…



```
standard.lib 4.10.05 11:14:46
Iecsfc.lib 13.4.06 15:51:28
Util.lib 18.5.10 15:14:28
SysLibTime.lib 26.3.12 16:06:01
SysTaskInfo.lib 26.3.12 16:06:01
SysLibMem.lib 26.3.12 16:06:01
SYSLIBCALLBACK.LIB 26.3.12 16:06:01
EtherCAT_AC500_V13.lib 26.3.12 16:06:27
Ethernet_AC500_V10.lib 26.3.12 16:06:29
BusDiag.lib 26.3.12 16:06:01
SysInt_AC500_V10.lib 5.4.12 08:45:24
SysExt_AC500_V10.lib 26.3.12 16:06:01
```

```
FUNCTION_BLOCK R_TRIG
(*
    Rising Edge detection.
*)
VAR_INPUT
    CLK: BOOL;   (* Signal to det
END_VAR
VAR_OUTPUT
    Q: BOOL;     (* Edge detectec
END_VAR
```

Right-click underneath the list of .lib files already included in the Library Manager and select 'Additional Library…'. A dialog will appear allowing you to navigate to new .lib files and include them by clicking on the 'Open' button (you can use the standard Windows CTRL and SHIFT keys to add multiple files as needed). Navigate to the directory where the PS552-MC-E library files have been installed (for Win7 systems this would typically be c:\Program Files(x86)\Common Files\CAA-Targets\ABB_AC500\AC500_V12\Library\PS552-MC).

Once you've located the PS552-MC-E library files include all of the following .lib files:

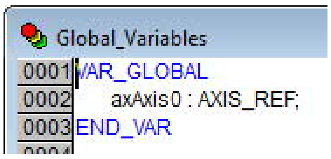
– CompactMotionControl_AC500_V12.lib

- ECAT_AC500_APPL_V21.lib
- MC_Base_AC500_V11.lib
- MC_Blocks_AC500_V11.lib

The library manager dialog will update to show these files are now included in the library for your project.

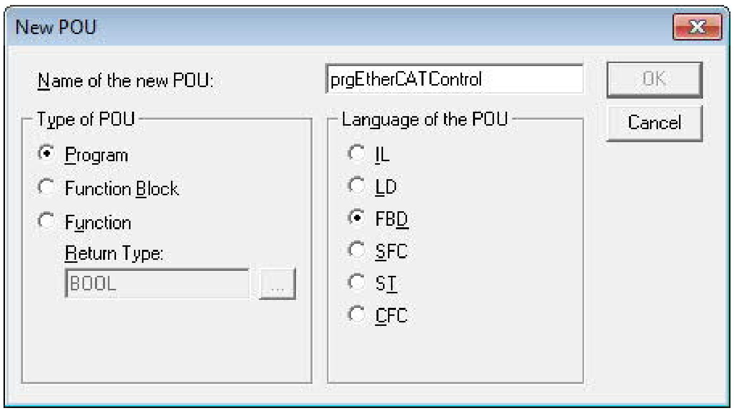
The next thing we need to do is create an axis structure that will be associated with our hardware device.

Double-click the 'Global Variables' icon on the 'Resources' tab to open the global variable editor window.
Declare a variable of type AXIS_REF (a pre-defined data structure for use with PLCopen motion control functions)….we named our variable 'axAxis0' to match the device name the axis is associated with and with the prefix defined by AN00244 (but you can call it anything you like)…


Global_Variables

```
0001 VAR_GLOBAL
0002     axAxis0 : AXIS_REF;
0003 END_VAR
```

Now click on the 'POU' tab of the CoDeSys project navigator. You will see that a default program has already been included (named PLC_PRG). For PLCopen motion projects using EtherCAT it is best to create a new program just for the core function blocks needed to control the axis. Whilst any of the IEC61131 languages may be used it is probably easiest to use Function Block Diagram (FBD) for a simple example. In the POUs tab, right click the POUs folder and select 'Add object'. Set the type of POU to Program and select FBD for the programming language. The POU can be named however you like, but for out example we will call this 'prgEtherCATControl' and we'll assume you've done the same…


New POU

Name of the new POU: prgEtherCATControl

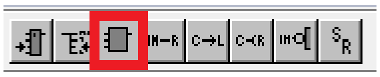
Type of POU: [✓ Program] Function Block Function

Return Type: BOOL

Language of the POU: IL LD [✓ FBD] SFC ST CFC

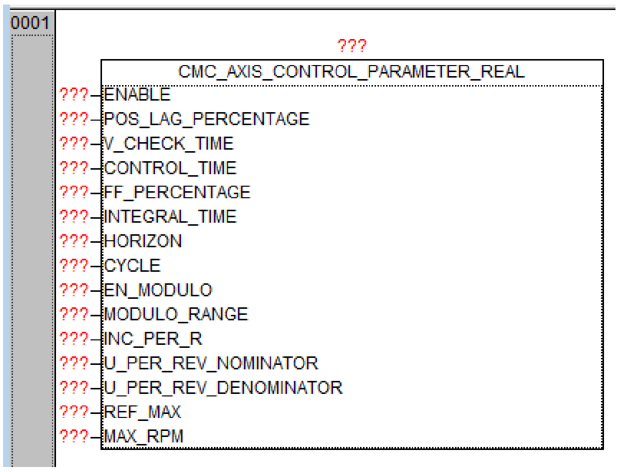
You can now right-click the PLC_PRG icon in the POU explorer and select 'Delete Object' to remove this program as it is no longer needed.

Double-click the prgEtherCATControl icon to open the code for this module. The program editor will show the variable declaration section for this code module at the top of the coding dialog and an empty rung 1 below. Click on the dotted empty box in rung 1 of the program and then click on the "Box" toolbar button at the top of the CoDeSys screen…

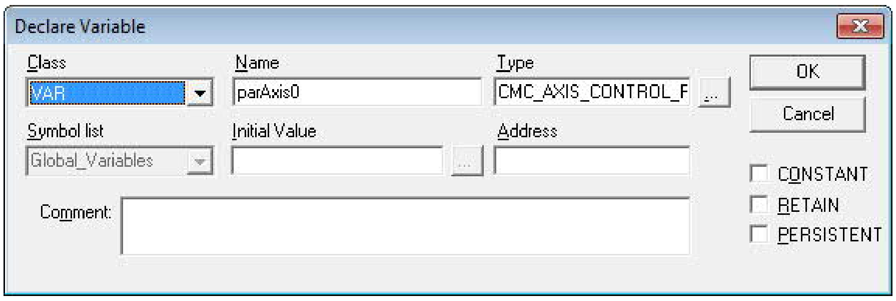


By default an FBD representation of the AND function will now appear in rung 1 of the program (with the AND text highlighted). We need to create a CMC_AXIS_CONTROL_PARAMETER_REAL function instance instead (used to setup axis scaling) so type in CMC_AXIS_CONTROL_PARAMETER_REAL (or Press F2 to use the Input assistant and select this block from the 'Standard Function Blocks' section) in place of the existing AND text…

If you've spelt this correctly, or picked it from the Input assistant correctly (and if the PS552-MC-E library files have been included correctly) then rung 1 will now display an instance of our control parameter function block…



Click on the "???" above the function block and type to give the instance of this function a name (we called ours parAxis0). After hitting "Return" on your keyboard CoDeSys will present a dialog asking you to declare the type for this variable (it will automatically select CMC_AXIS_CONTROL_PARAMETER_REAL so you can just click OK)…


Declare Variable

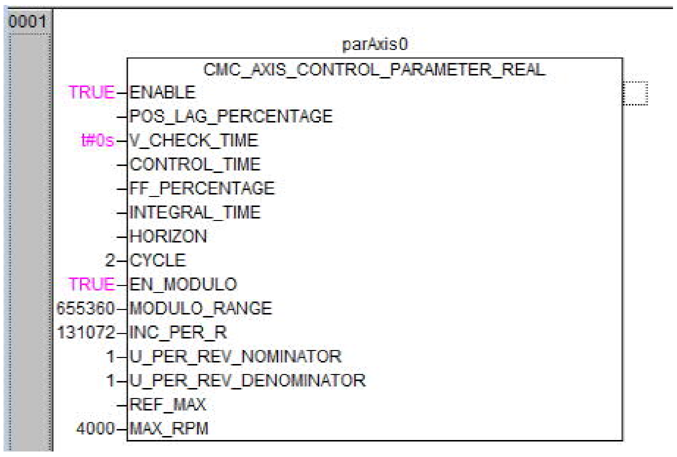
Class: VAR
Name: parAxis0
Type: CMC_AXIS_CONTROL_F
Symbol list: Global_Variables
Initial Value:
Address:
Comment:
CONSTANT
RETAIN
PERSISTENT

The table below shows the input parameter values that must be entered (all other input parameters must be deleted – i.e. select the ??? and delete it)…

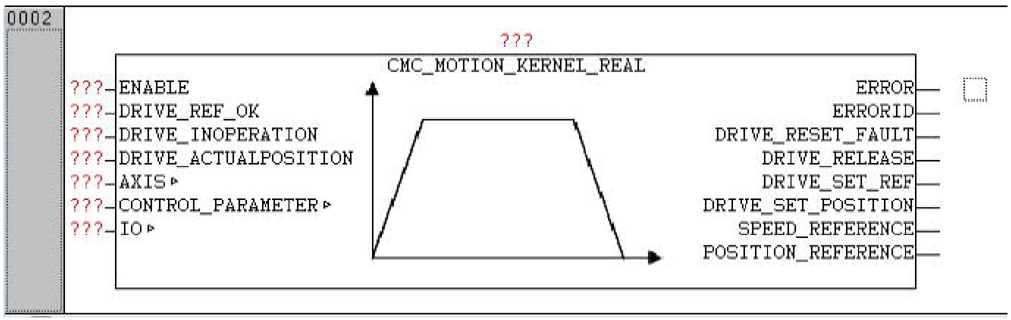
Input parameter	Description	Value	Data type
ENABLE	Determines whether the PLC should process the axis every bus cycle or not (must be set FALSE in order to modify scaling values but for this example we will use fixed scaling so can set it TRUE permanently)	TRUE	BOOL
V_CHECK_TIME	Used to set the sample time for deriving measured speed (not really used when using EtherCAT but must currently be forced to zero)	t#0s	TIME
CYCLE	Sets the bus cycle time – this must match the EtherCAT bus cycle time we set earlier via Automation Builder	2	LREAL
EN_MODULO	This parameter should be set TRUE if the axis is to be treated as a modulo axis (i.e. there a wrap on the axis position at a predefined range – this can be useful for rotary turntables/tool-changers where absolute position within a cycle can then be determined) or if the axis position will wrap past the 32 bit position boundary (e.g. a continuously rotating axis). If the axis only moves forwards/backwards within the 32 bit position range and there is no requirement for modulo functionality then set this parameter FALSE. For this application note we will set it TRUE (as we could keep moving forwards indefinitely)	TRUE	BOOL
MODULO_RANGE	Used in combination with EN_MODULO. Sets how many encoder counts there are in one modulo cycle. For our example we'll assume our 131072 count motor is fitted to a 5:1 gearbox so we'll set MODULO_RANGE to 655360 (therefore reading position will return the absolute position of the gearbox output). If you aren't interested in absolute position within a cycle (e.g. if this was a continually running conveyor) then set MODULO_RANGE to 2147483647	655360	DINT
INC_PER_R	Sets how many encoder counts the drive receives for each rev of the connected motor (for this	131072	DWORD

	example we've got a BSM60R-240MT motor connected to our drive so there are 131072 counts in one motor rev – set your parameter to suit your motor)		
U_PER_REV_NOMINATOR	This parameter (in combination with the denominator parameter) allows you to configure the axis scaling. Note that the term "NOMINATOR" is more accurately a "NUMERATOR" in mathematics. You must specify how many user units there are in 1 rev of the motor (using the numerator and denominator as necessary to represent fractional values). For our example we'll use a user unit of "Revs" so there is 1 user unit in 1 rev of the motor (hence numerator and denominator can both be set to 1)	1	DINT
U_PER_REV_DENOMINATOR	See above	1	DINT
MAX_RPM	"Calibrates" the PLC to suit the "DRIVESPEEDMAX" configured on the drive (check the DRIVESPEEDMAX setting on the drive via Workbench if you don't already know the maximum application speed setup on the drive – or use the Operating Mode pages of the Workbench to determine the application max speed in rpm). For our example we had a drive configured for a maximum speed of 4000 rpm	4000	WORD

Having entered these values the rung should look like this…


```
0001
                    parAxis0
          CMC_AXIS_CONTROL_PARAMETER_REAL
   TRUE-ENABLE
       -POS_LAG_PERCENTAGE
   t#0s-V_CHECK_TIME
       -CONTROL_TIME
       -FF_PERCENTAGE
       -INTEGRAL_TIME
       -HORIZON
      2-CYCLE
   TRUE-EN_MODULO
 655360-MODULO_RANGE
 131072-INC_PER_R
      1-U_PER_REV_NOMINATOR
      1-U_PER_REV_DENOMINATOR
       -REF_MAX
   4000-MAX_RPM
```

At this point it is worth saving the PLC program (click on the floppy disk icon at the top of CoDeSys to save the file).
Right-click in the rung 1 grey area and select 'Network (after)' to add a second rung to the PLC code. Again click on the dotted box (this time in rung 2) and then click on the "Box" icon in the toolbar to insert another function block. This time type CMC_MOTION_KERNEL_REAL or use the Input assistant again (this will add an instance of the floating point motion profiler). If you've done this correctly rung 2 will look like this…

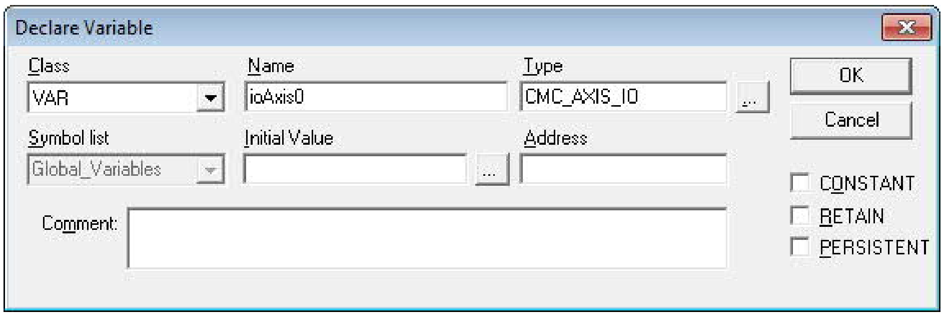

```
0002
                              ???
                     CMC_MOTION_KERNEL_REAL
???-ENABLE                                         ERROR-
???-DRIVE_REF_OK                                 ERRORID-
???-DRIVE_INOPERATION                   DRIVE_RESET_FAULT-
???-DRIVE_ACTUALPOSITION                    DRIVE_RELEASE-
???-AXIS ▹                                  DRIVE_SET_REF-
???-CONTROL_PARAMETER ▹                DRIVE_SET_POSITION-
???-IO ▹                                  SPEED_REFERENCE-
                                       POSITION_REFERENCE-
```

Click on the ??? above the function block to name this instance (we called ours kerAxis0). Again CoDeSys will present a dialog allowing you to specify the variable type and will automatically make the variable a CMC_MOTION_KERNEL_REAL type so just accept this.
Again there are some input parameters to this function block that we must enter. Also, the profiler output must be directed to the drive. As we have earlier configured the drive for "Cyclic Synchronous Position" operation it's the "POSITION_REFERENCE" output of the Kernel that must be targeted at the drive. The table below shows which parameters must be entered (just delete any ??? which aren't needed). Most of the output parameters aren't needed but it might be useful to assign these outputs to variables for diagnostics. If you decide to assign variables to some of these (e.g. ERRORID) then you can determine the

variable type required by right-clicking on the function block and selecting "ZOOM" to see a declaration for the block which includes details on the parameter variable types.

Input parameter	Description	Value	Data type
ENABLE	Determines whether the PLC should process the profiler every bus cycle or not so can set TRUE permanently	TRUE	BOOL
DRIVE_ACTUALPOSITION	Feedback of the position (in counts) of the motor connected to the drive. This is one of the PDO objects that we setup earlier in Automation Builder and we included a mapped variable for this – diAxis0ActualPos	diAxis0ActualPos	DINT
AXIS	Data structure for the axis to be profiled. In this case we use the AXIS_REF type variable we declared earlier – axAxis0	axAxis0	AXIS_REF
CONTROL_PARAMETER	Reference to the control parameter block being used by the axis. Here we enter the name we gave our CMC_CONTROL_PARAMETER_REAL block earlier. This links the profiler and axis parameters together	parAxis0	CMC_CONTROL_PARAMETER_REAL
IO	Data structure for some of the IO associated with the axis (e.g. home sensor, positive and negative limit switches)	ioAxis0	CMC_AXIS_IO (see note below)

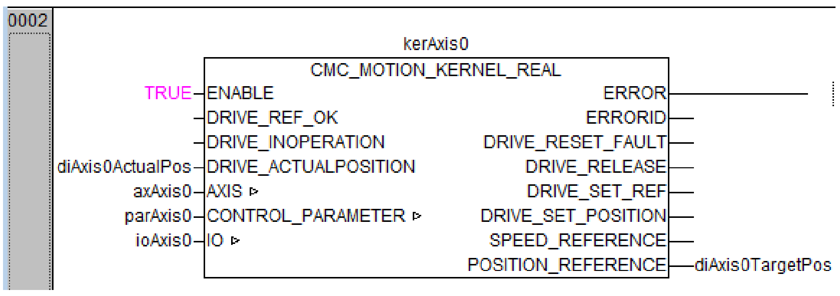
Note: When adding ioAxis0 as the name of the IO input parameter CoDeSys will ask for the data type for this variable (as it is a new variable so far undeclared). Enter CMC_AXIS_IO for the data type as shown below – this is a pre-defined data type to suit this Axis I/O structure…


Declare Variable

Class: VAR	Name: ioAxis0	Type: CMC_AXIS_IO	OK / Cancel
Symbol list: Global_Variables	Initial Value:	Address:	☐ CONSTANT ☐ RETAIN ☐ PERSISTENT
Comment:			

Output parameter	Description	Value	Data type
POSITION_REFERENCE	Position target output by the PLC profiler that must be sent to the drive. This is one of the PDOs we mapped earlier and we included a mapped variable for this – diAxis0TargetPos	diAxis0TargetPos	DINT

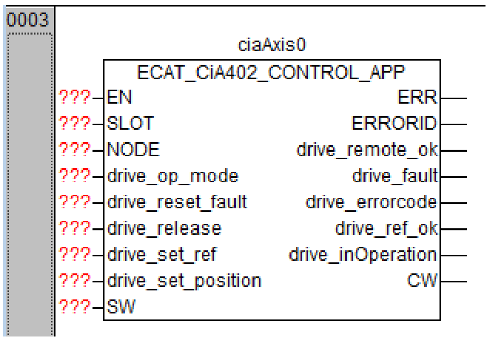
Having entered these values the rung should look like this…



Now add a third rung and this time add a box of type ECAT_CiA402_CONTROL_APP (we named ours ciaAxis0 and as you'd expect, it's a variable of type ECAT_CiA402_CONTROL_APP).

Note: The ECAT_CiA402_CONTROL_APP block is used for both MicroFlex e190 and MotiFlex e180 programming. There is also an older block named ECAT_CiA402_CONTROL_e150_APP that will also work for both drive types but it is best to select the one recommended above.

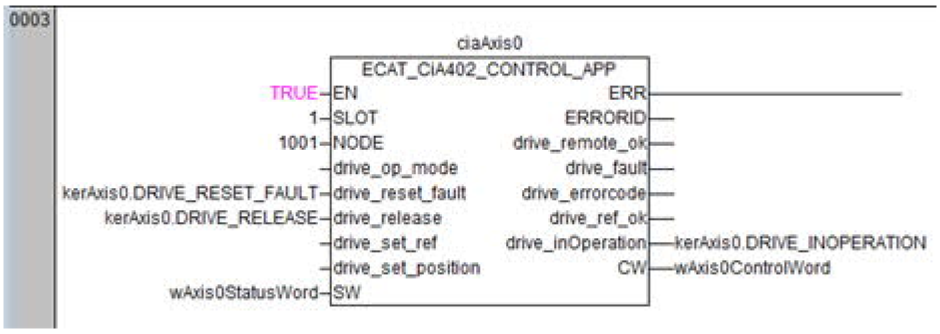
This is a function block that encapsulates the CiA DS402 drive control profile for the MicroFlex e190 and MotiFlex e180 drives and allows this profile to operate via EtherCAT. We have to link this function block to the drive itself (via the drives EtherCAT address which, we saw earlier, was set to 1001 by Automation Builder automatically). We also have to link this function block to the Kernel for the axis (via some of the Kernel's parameters). Lastly we have to relate the relevant DS402 Controlword and Statusword to this function block. These were both PDOs (to/from the drive) for which we included mapped variables earlier.



The table below shows the required input and output parameters for our Axis0_DS402 function block:

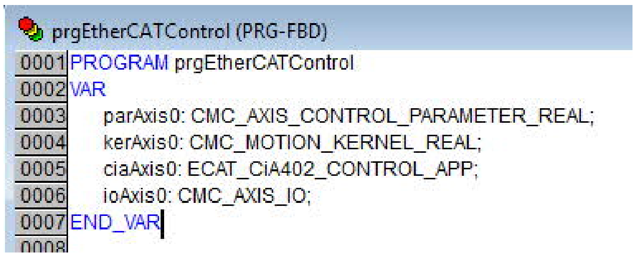
Input parameter	**Description**	**Value**	**Data type**
EN	Determines whether the PLC should process the DS402 PDOs every EtherCAT cycle. For our example we're expecting to control the drive at all times so we'll set this to TRUE permanently	TRUE	BOOL
SLOT	Which slot is the CM579-ECAT module plugged into? In this case we plugged it into the first slot to the left of the processor so it's slot 1. For PM595 this value is 5 if using ETH3 or 6 if using ETH4	1	BYTE
NODE	The EtherCAT address automatically assigned by Automation Builder when we added the drive (1001)	1001	DWORD
Drive_reset_fault	Link to the fault reset request from the associated axis kernel	kerAxis0.DRIVE_RESET_FAULT	BOOL
Drive_release	Link to the drive release request from the associated axis kernel	kerAxis0.DRIVE_RELEASE	BOOL
SW	The DS402 status word from the associated drive (we mapped the wAxis0StatusWord variable to this drive PDO earlier)	wAxis0Statusword	WORD
Output parameter	**Description**	**Value**	**Data type**
Drive_inOperation	Output parameter that links to the DRIVE_INOPERATION input parameter of the associated axis kernel	kerAxis0.DRIVE_INOPERATION	BOOL
CW	The DS402 control word to the associated drive (we mapped the wAxis0ControlWord variable to this drive PDO earlier)	wAxis0ControlWord	WORD

Having entered these values the rung should look like this…



As with the previous rung, most of the output parameters aren't needed but it might be useful to assign these outputs to variables for diagnostics. If you decide to assign variables to some of these (e.g. drive_errorcode) then you can determine the variable type required by right-clicking on the function block and selecting "ZOOM" to see a declaration for the block which includes details on the parameter variable types.

If you have decided not to assign any additional variables to the three function blocks added then you should at this point only have the automatically added variable declarations…


prgEtherCATControl (PRG-FBD)

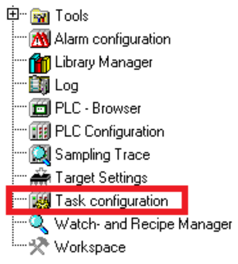
```
PROGRAM prgEtherCATControl
VAR
	parAxis0: CMC_AXIS_CONTROL_PARAMETER_REAL;
	kerAxis0: CMC_MOTION_KERNEL_REAL;
	ciaAxis0: ECAT_CiA402_CONTROL_APP;
	ioAxis0: CMC_AXIS_IO;
END_VAR
```

Again it's probably worth saving the PLC program (and the Automation Builder project) at this point.

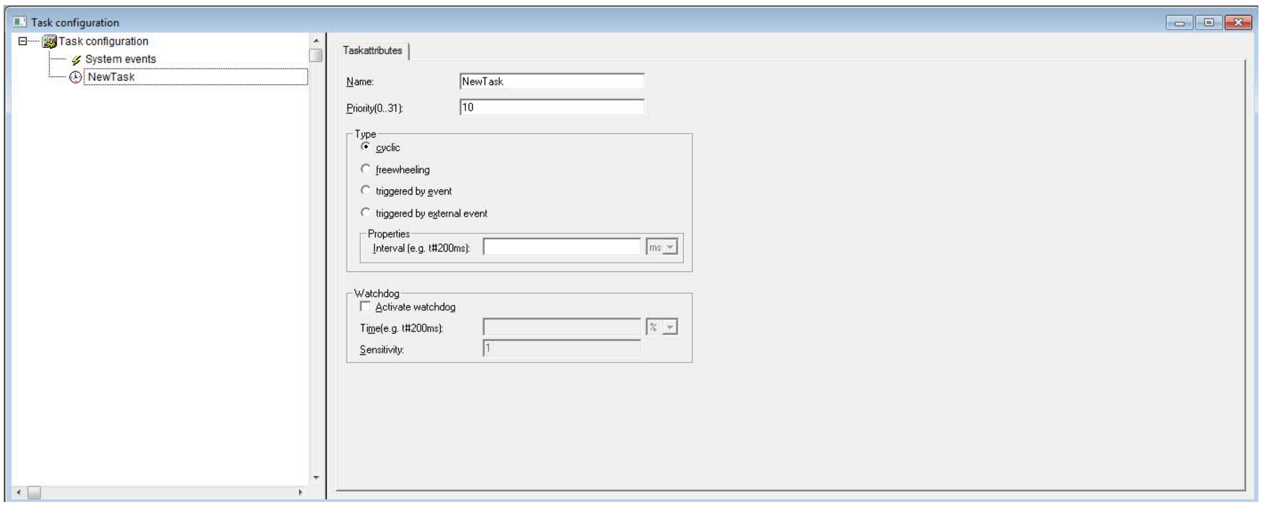
The three rungs we've added so far form the "core" for control of the ABB servo drive via EtherCAT, we're now ready to call this program from a PLC Task and ensure it is synchronised with data transfer to/from the EtherCAT network.

Task Configuration

Click on the 'Resources' tab in the CoDeSys project navigator. Then double-click the 'Task configuration' icon…

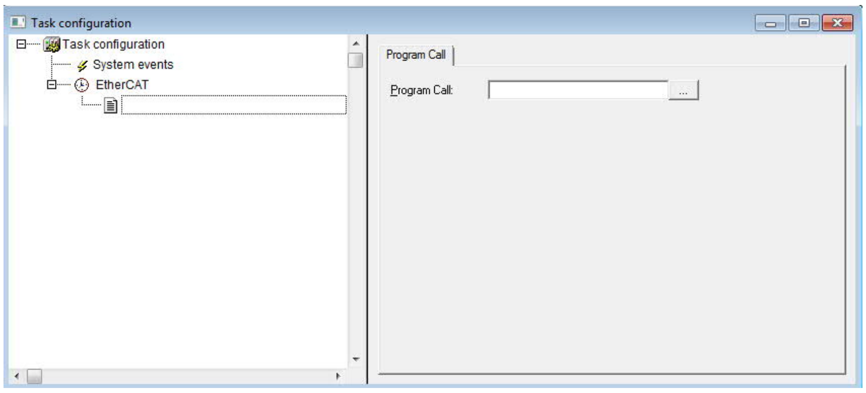


…the right hand pane in CoDeSys will now display the Task Configuration dialog. Right-click the Task Configuration icon in the right hand pane's dialog and select 'Append Task'. The dialog should look like this…

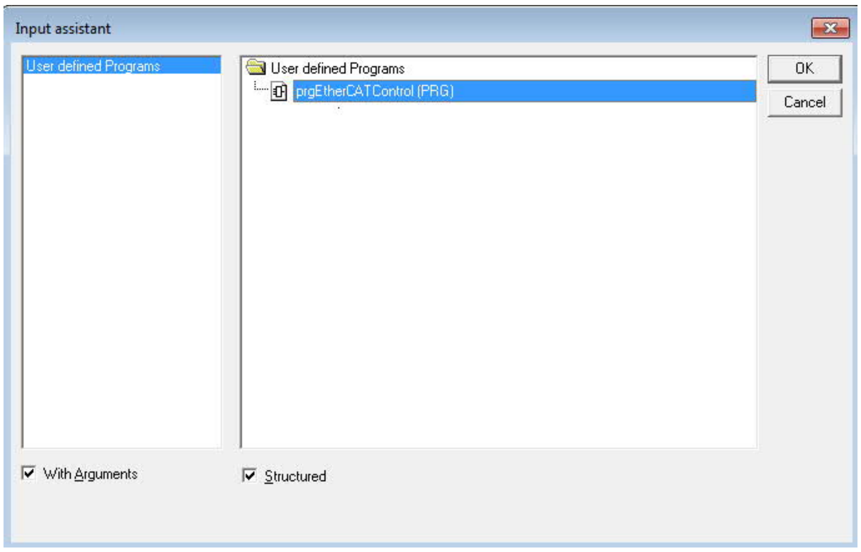


Name: NewTask
Priority(0..31): 10
Type: [✓ cyclic] freewheeling / triggered by event / triggered by external event
Properties: Interval (e.g. t#200ms): ms
Watchdog: ☐ Activate watchdog
Time(e.g. t#200ms): %
Sensitivity: 1

We need to configure a task that executes our program (prgEtherCATControl) in relation to events generated by the EtherCAT coupler. Click on the "NewTask" text and rename this to read "EtherCAT". Now right click the EtherCAT icon and select 'Append Program Call'. At this point your task configuration dialog will look like this…

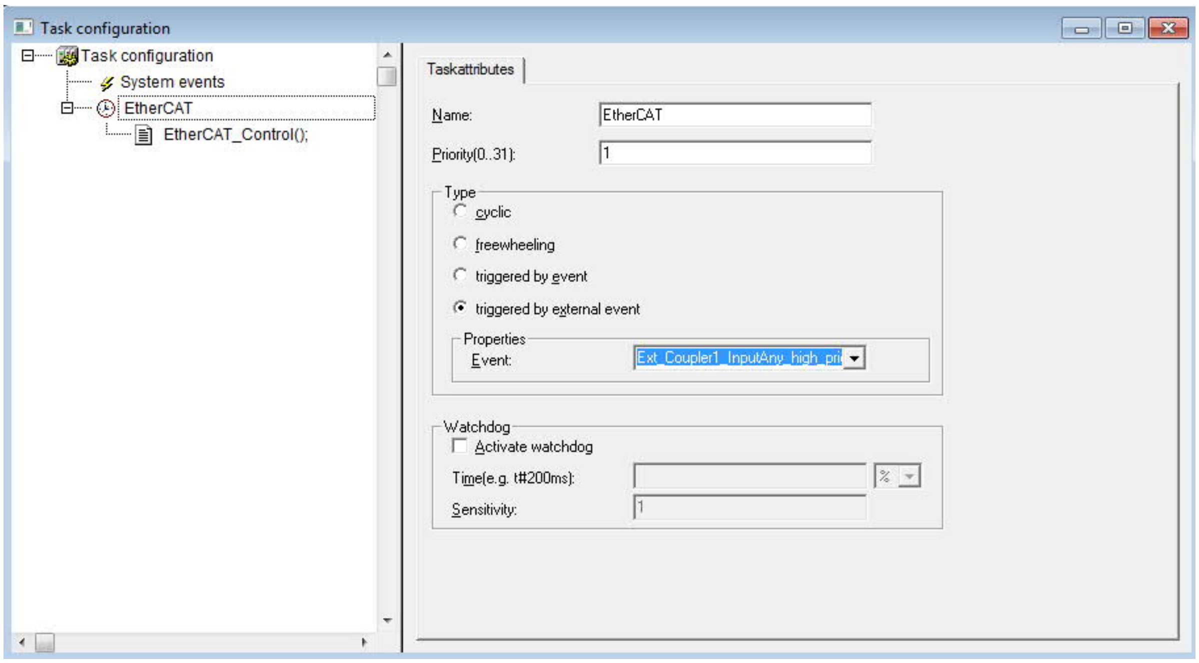


In the right hand pane click on the button with "…" on it and select prgEtherCATControl (the only user program we have in our project)…



Click on OK.

We now need to setup the EtherCAT task so that it is triggered by an external event (i.e. ensure that processing of the task is related to the EtherCAT cycle). Click on the 'EtherCAT' icon and in the right hand pane first set the Priority to 1 and then select the "triggered by external event" option.
The event properties dropdown will then provide a series of options to select from….click on 'Ext_Coupler1_InputAny_high_priority' (or Ext_Coupler5_InputAny_high_priority if you are using a PM595 and have setup ETH3 to act as the EtherCAT coupler).



Name: EtherCAT
Priority(0..31): 1
Type
cyclic
freewheeling
triggered by event
[✓ triggered by external event]
Properties
Event: Ext_Coupler1_InputAny_high_pri
Watchdog
Activate watchdog
Time(e.g. t#200ms):
Sensitivity: 1

By selecting 'Ext_Coupler1_**InputAny**_high_priority' we are ensuring that the EtherCAT task is processed after *reception* of every EtherCAT telegram which results in the shortest possible reaction time. 'Ext_Coupler1_**Input2Any**_high_priority allows the task to be processed when the EtherCAT telegram is *sent* which produces a reaction time 1 cycle longer. For each input type there is also a normal priority version of the event which places a lower priority on the execution of the associated task but for motion control it is best to always use the highest priority event.

You can now close this dialog and save the PLC program in CoDeSys (and also save the Automation Builder project). You can now try selecting Project > Build from the CoDeSys menu options…..if you have followed all the instructions correctly so far the project should build with 0 errors and 0 warnings. If this isn't the case please re-read the previous pages and check your work so far.

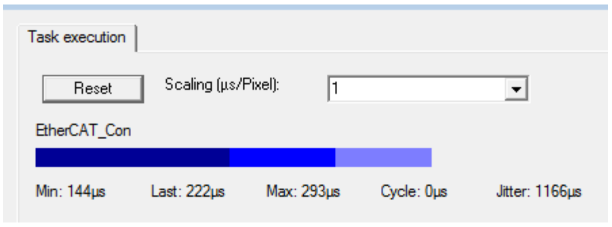
If the program builds with no errors then we are ready to start adding some simple motion commands to the project.

Adding simple motion

So far the motion code we've written has been restricted to the basic function blocks needed to control our EtherCAT axis. These blocks should always be in a POU called by the task triggered from the EtherCAT coupler event (we'll call this the 'real time POU' for the rest of this document). Other examples of code/logic that should be included in this real-time POU would be….

- Touchprobe function blocks
- Home to Touchprobe function blocks
- Logic to detect axis position or encoder thresholds very accurately
- Motion blocks that need to be triggered by axis positions or encoder values being reached
- Function blocks relating to processing of a master encoder via a CD522 encoder module

For many applications (and this quick start guide) it is not necessary to include the general motion logic in the real time POU. The more code that is included in the real time POU the longer the processing time for this code. It is important that the processing time for the real time POU is less than or equal to the EtherCAT cycle time minus 300 µs - this can be monitored when Online to the running PLC program via the 'Task Configuration' dialog in CoDeSys…

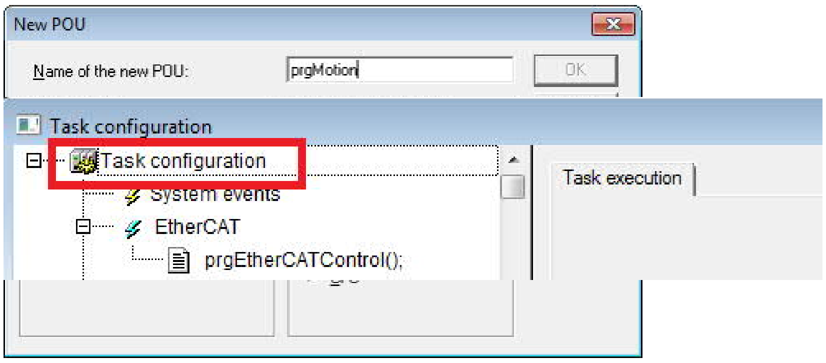


Scaling (µs/Pixel): 1
EtherCAT_Con
Min: 144µs Last: 222µs Max: 293µs Cycle: 0µs Jitter: 1166µs

In this snapshot from an application the maximum execution time for the real time POU is 293 µs. We must add to this the 'Jitter' in the execution. There is a bug in CoDeSys that adds 1000 to the displayed Jitter (it doesn't show the actual value for the available Cycle either) so in our example above the total execution time (worst case) for our POU is 293 µs + 166 µs = 459 µs. This is well within our 2000 µs – 300 µs (i.e. 1700 µs) limit so there is plenty of scope to add additional code to the real time POU if we needed to.

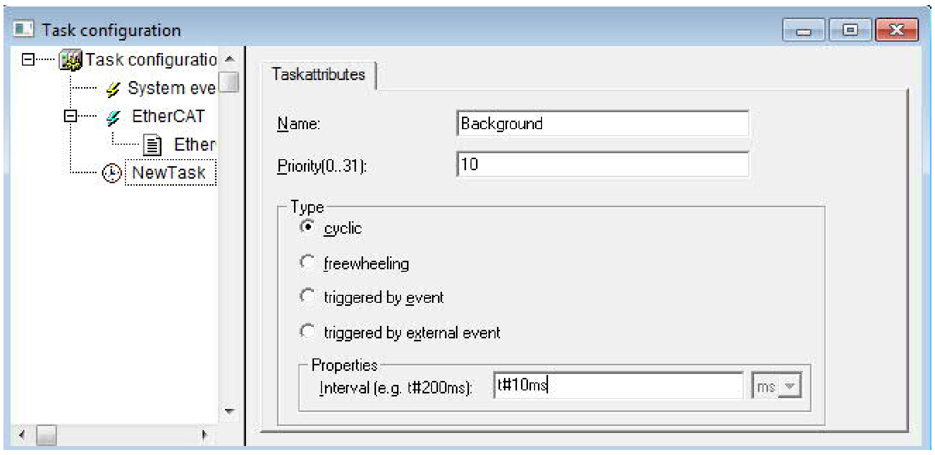
For now we will add a new Task to the Task Configuration and call a new POU/program from this task. This new task can be Cyclic. For our example we will call this task ‘Background’ and we will call a Program named ‘prgMotion’ from this task.

In the POU explorer, right-click the POUs folder and select ‘Add object’. We will add the ‘Motion’ program as an item programmed using Function Block Diagram (FBD) as this is very commonly used for very simple applications. Configure the resulting dialog as shown below…



Click OK and this program unit will be added to the POU tree. Now switch to the Task configuration dialog (the window may still be open in the editor somewhere if you have not closed it or you may need to double-click ‘Task configuration’ on the ‘Resources’ tab again. Right-click the ‘Task configuration’ icon in the dialog…

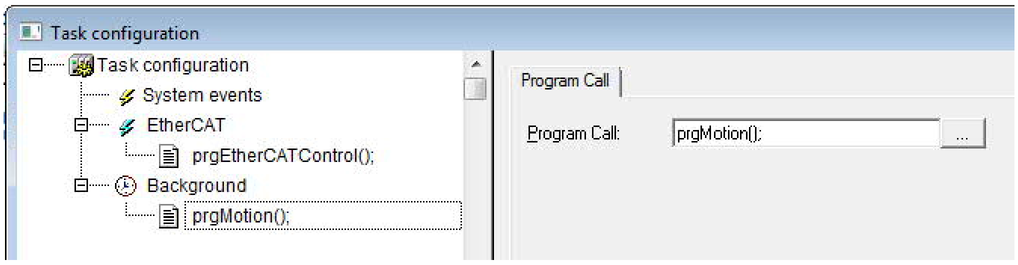
…and select ‘Append Task’ – a new task will appear in the tree. Select this new task and then edit the dialog as shown below to create a task named ‘Background’ that is Cyclic with a cycle time of 10 ms…



Name: Background
Priority(0..31): 10
Type
(● cyclic) freewheeling / triggered by event / triggered by external event
Properties
Interval (e.g. t#200ms): t#10ms ms

Note that the name of the task will change from ‘NewTask’ to ‘Background’ after closing this dialog or clicking somewhere else in the tree.

We now need to add the call to our Motion program, so right click the ‘NewTask’ icon (or ‘Background’ icon if you have already forced the name to change) and select ‘Append Program Call’. Click on the button with three periods (…) and use the Input assistant (F2 keyboard shortcut) to select the Motion program unit or type in the name of the program (i.e. prgMotion();) as shown below…



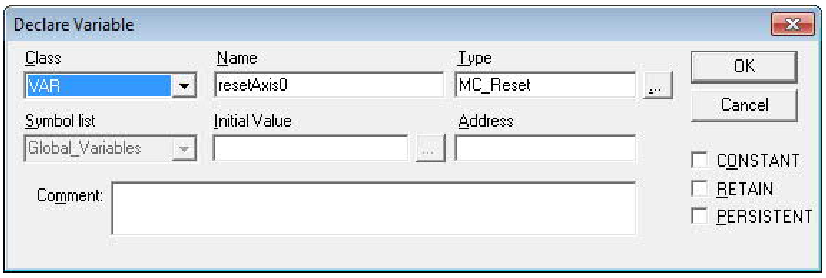
Program Call: prgMotion();

You can now close this dialog, we are ready to start adding motion blocks to the motion program. Switch back to the POU explorer and double-click the ‘prgMotion’ program icon to open the program. You will see a blank network 0001 ready to accept code. By now you should know how to add function blocks to the program (select a new box, use F2 / Input assistant to select the function block name or type in the name etc…) so we will just list the function blocks you need to add to the code and describe the necessary input and output parameters. You can add these blocks in any order for this example, but we’ll assume that you add them in the order they are described below.

MC_Reset

If the drive is in a fault state it is not possible to enable the axis and/or demand any motion. Adding this function block will allow us to reset any drive errors that may have occurred (e.g. “following error” resulting from demanding excessive acceleration or “PDO data missing” resulting from switching the e180 drive to direct mode for fine tuning for example). If the drive is in an error condition the fault code will be displayed on the seven segment display (e.g. 1 0 0 0 5 for following error). If the PLC program is running the drive_errorcode output of the Axis0_DS402 function block will also indicate the fault code.

Click on the dashed box outline in network 0001 if this is the first block/network you’re adding (or right-click the last rung in your program and select ‘Network (after)’ to add another rung). Add a MC_Reset function block and give it a relevant name (we called ours resetAxis0). CoDeSys will ask you to declare this variable (i.e. the instance of the MC_Reset function block you just


Declare Variable

Class: VAR
Name: resetAxis0
Type: MC_Reset
Symbol list: Global_Variables
Initial Value:
Address:
Comment:

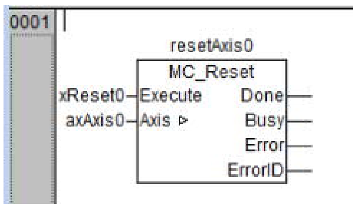
named). A dialogue will appear showing the Class as ‘VAR’, Name as ‘resetAxis0’ (or whatever you called your block) and Type as ‘MC_Reset’ automatically as shown below…

Just click OK to accept this and the variable declaration will be added to the top of the program file.

This PLCopen function block only takes two inputs parameters:

- Execute : a rising edge on this input causes the function block to process the actions necessary to try and reset errors on the servo drive. A BOOL type variable should be assigned to this input (we named ours xReset0)….simply click on the ??? next to the Execute input and type/enter the name of the variable….CoDeSys will display a dialog allowing you to configure the scope and data type (which will be BOOL by default so you can just acknowledge this dialog by clicking OK)
- Axis : this is the axis structure relating to the function block (in our case axAxis0)

For this simple example we didn’t bother assigning variables to any of the MC_Reset output parameters. Here’s how our completed rung looked…



When we come to test our program later we will double-click the input variable (xReset0) and then press CTRL+F7 to force the variable value we’ve selected (e.g. TRUE to reset an error).

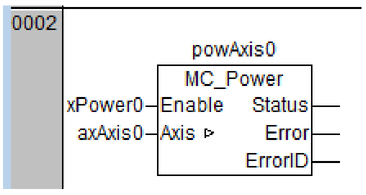
MC_Power

This PLCopen function block allows the user to enable and disable the associated axis. Right-click the last rung in your program and select ‘Network (after)’ to add another rung. This time add a MC_Power function block and give it a relevant name (we called ours ‘powAxis0’). Every time you create a variable, as we’ve already seen, the Declare Variable dialog will appear and unless we mention otherwise you can just click OK to create the variable declaration.

This PLCopen function block only takes two inputs parameters:

- Execute : a rising edge on this input causes the function block to process the actions necessary to try and enable the axis. A falling edge on the input causes the function block to disable the axis. A BOOL type variable should be assigned to this input (we named ours xPower0)
- Axis : this is the axis structure relating to the function block (in our case axAxis0)

Here's how our additional rung looked after adding the MC_Power function block…



For this simple example we didn't bother assigning variables to any of the MC_Power output parameters, but it can be useful to use the Status output (powAxis0.Status in our example) as an interlock on motion functions (i.e. only allow motion to be commanded if the status indicates the drive is enabled).

When we come to test our program later we will double-click the input variable (xPower0) and then press CTRL+F7 to force the variable value we've selected (e.g. TRUE to enable the drive).

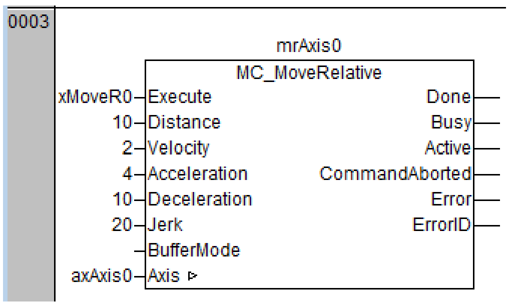
MC_MoveRelative

This PLCopen function block allows the user to move the axis a relative distance from its current position. Right-click the last rung in your program and select 'Network (after)' to add another rung. This time add a MC_MoveRelative function block and give it a relevant name (we called ours 'mrAxis0').

This PLCopen function block takes several inputs parameters:

- Execute : a rising edge on this input causes the function block to process the actions necessary to try and move the specified axis by the defined distance. A BOOL type variable should be assigned to this input (we named ours xMoveR0)
- Distance : the amount of movement required (+/-) when the execute input is activated. You could define a variable for this input to make the distance adjustable but for our example we "hard coded" a distance of 10 units (you may recall earlier we used the U_PER_REV_NOMINATOR and U_PER_REV_DENOMINATOR input parameters to our Control Parameters function block to set our user units equal to motor revolutions)
- Velocity : the slew speed for the move in user units per second (in our case revs/second). You could define a variable for this input to make the velocity adjustable but for our example we "hard coded" a speed of 2 revs/second
- Acceleration : the acceleration rate for the move in user units per second per second (in our case revs/second^2). You could define a variable for this input to make the acceleration adjustable but for our example we "hard coded" an acceleration rate of 4 revs/second^2
- Deceleration : the deceleration rate for the move in user units per second^2 (in our case revs/second^2). You could define a variable for this input to make the deceleration adjustable but for our example we "hard coded" a deceleration rate of 10 revs/second^2
- Jerk : the rate of change of acceleration/deceleration in user units per second^3. Adding a value for Jerk results in a "s-ramped" velocity profile. Setting Jerk to 0 results in a trapezoidal velocity profile. You could define a variable for this input to make the amount of Jerk adjustable but for our example we "hard coded" a value of 20 revs/second^3
- BufferMode : the ABB motion libraries do not currently support any buffer mode other than mcAborting (where mcAborting is a pre-defined constant contained within the library). For our example we will delete the ??? characters for this input parameter, setting BufferMode to its default setting of mcAborting – i.e. non-buffered
- Axis : this is the axis structure relating to the function block (in our case axAxis0)

Here’s how our additional rung looked after adding the MC_MoveRelative function block…



We didn’t add any of the output parameters but these can be included if required (e.g. variables could be assigned to the ‘Done’ output to indicate when the move is complete or to the ‘ErrorID’ output to indicate the fault code if the move fails).

We’ve added enough code now to allow enabling/disabling of the drive and for a simple relative move to be performed.

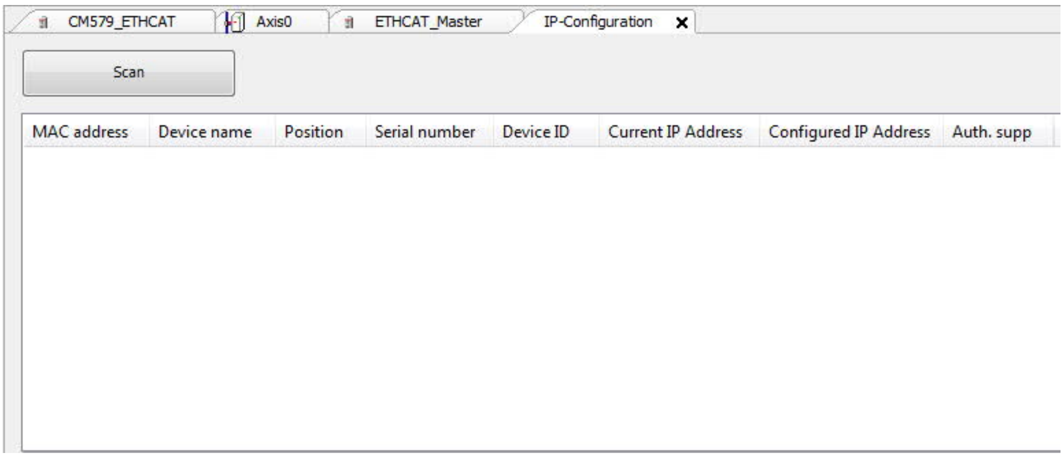
Testing the PLC program

To test the PLC program it is assumed the drive has been commissioned and ready to be enabled (e.g. AC power is applied to the drive, STO inputs are inactive) – please refer to the drive's installation manual for further details if needed.

If you've not been online to the PLC before, select the Online>Communication Parameters menu in CoDeSys and ensure you've selected the PLC you're using (if you need to add a new entry select 'New…', select the 3S TCP/IP device, enter the IP address for your PLC, select Port 1201 and Motorola byte order).

If you're not sure what IP address your PLC is setup for then switch to Automation Builder and select the Tools>IP Configuration menu. Note, you will need to disable any firewalls you have configured to allow this tool to work.

The right hand pane will then show the following…



MAC address	Device name	Position	Serial number	Device ID	Current IP Address	Configured IP Address	Auth. supp

Click 'Scan'. This will scan your Ethernet port(s) and attempt to find connected PLCs (and all ABB MAC addresses connected to the network). The resulting dialog will indicate the IP address of the PLC (and will allow you to change it if you like). You will need to ensure your PC's network adaptor is on the same subnet (e.g. If the PLC is found at 192.168.0.10 then your PC adaptor needs to be 192.168.0.x , where x is a value other than 0, 10 or 255 and not used by any other Ethernet devices on the same network).

Once the CoDeSys communication parameters are correct, Compile your project (Project > Build) and look to the dialog box at the bottom of the screen to check there are no errors and that it compiles correctly. If CoDeSys reports any errors you will need to resolve these before the project will download successfully – refer to the PLC programming documentation for further assistance if needed.

Once the project has downloaded select the Online>Run menu option….the PLC program should now run (the front of the PLC CPU should display RUN, the bottom right hand corner of CoDeSys should indicate 'Running' and as the PLC transfers the EtherCAT configuration to the drive the green Network status LED on the drive should flash (turning solid once configuration has completed and the drive has entered operational status).
If the drive seven segment is displaying "-" then there are no errors present on the drive and we can attempt to enable it. If the drive seven segment is flashing some sort of error code (a series of digits that repeat in sequence) then we will need to reset the error.

Scroll down through the PLC program until you find the rung containing the MC_Reset function block. Double-click the 'xReset0' variable…each time you double-click it CoDeSys will show/toggle a force value adjacent to the variable name (in this case, as the variable is of BOOL type the force value will toggle from FALSE to TRUE and vice-versa). Set the force value to TRUE and then press CTRL+F7 to write this value to the PLC. The rising edge (from FALSE to TRUE) on the MC_Reset function block's 'Execute' input should cause any faults on the drive to be reset (if an error code was flashing previously this should be replaced by a solid dash).

Repeat this process but force the value of xReset0 back to FALSE.

If you are unable to reset a drive error, connect to the drive using Workbench and analyse the cause of the error further (e.g. maybe the error cannot be reset for some logical reason).

Once the drive is displaying a dash, scroll down through the PLC program to the rung containing the MC_Power function block.

This time double-click the xPower0 variable until it shows TRUE and press CTRL+F7 to write this to the PLC. If all is well the drive should enable (and the seven segment display should indicate 8 or P depending on firmware version being used on the drive).
Leave xPower0 set to TRUE (setting it to FALSE will cause the drive to disable).

Scroll down through the PLC program until you find the rung containing the MC_MoveRelative function block. Double-click the xMoveR0 variable until it shows TRUE and press CTRL+F7 to write this to the PLC. If all is well, every time the value of bMoveR0 changes from FALSE to TRUE the PLC should profile a relative move (of 10 revs if you've followed this application note exactly).

If you scroll to the top of the program (to the variable declaration area) you can expand the definitions for things like the mrAxis0 function block and examine the state of the output parameters (e.g. the Done and Busy outputs). Select the Resources tab, expand the Global variables tree to find the Axis0_Module_Mapping variables and you can examine the PDO data being passed between the PLC and the drive.

If you expand the Global variables module itself you'll find the definition for axAxis0 as an AXIS_REF and you can expand this to view the full range of data available for the axis.

Congratulations! You've successfully controlled an ABB servo drive from an AC500 PLC via EtherCAT. Please refer to the documentation for the PS552-MC-E libraries for further information.

Additional resources

Now you are familiar with the basic operation of ABB servo drives with the AC500 PLC over EtherCAT you may want to explore some additional topics. You will find further application notes to support EtherCAT operation on the motion support website… http://new.abbmotion.com/support/SupportMe/ApplicationNotes.asp).

At the time of writing this particular document the following application notes are also available:

AN00203 – Using TwinCAT
AN00220 – EtherCAT homing methods
AN00221 – EtherCAT fast position capture
AN00234 – Generic Drive Interface via EtherCAT for simple motion
AN00239 – Using the CD522 module for master encoder input
AN00240 – Using the CD522 module fast latch inputs
AN00241 – Using the motion drive encoder channels for master encoder input
AN00242 – Accessing drive parameters via EtherCAT SDO
AN00243 – Initialising an EtherCAT network
AN00244 – PLC coding style guidelines
AN00245 – Linear flying shear example
AN00246 – Indexing conveyor example
AN00252 – Accessing drive error data via EtherCAT
AN00253 – Rotary knife example

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