Motion Control Products

Application note
Getting started with AC500 v3 motion

AN00268

Rev A

Motion in minutes using pre-written PLCopen XML files that can be imported into an Automation Builder AC500 v3 PLC project

Introduction

This application note details how to use Automation Builder for v3 AC500 PLCs (AB v2.2.2 onwards) 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 v3 PLC program to perform motion on these drives. A pre-written import file is used to create an axis “class” that includes all the basic functions any motion axis is likely to require.

Pre-requisites

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

- Mint Workbench build **5852** or later (see www.abb.com/motion for latest downloads and support information)
- A MicroFlex e190 or MotiFlex e180 drive with build **5900** or later firmware
- A PC or laptop running Automation Builder **2.2.2** or later
- An installed (and licensed) copy of the ABB PLCopen v3 motion control library (prior to release this is distributed as a number of license free components that are included with this document)
- One of the following AC500 PLC processors…..PM5630-2ETH, PM5650-2ETH, PM5670-2ETH, PM5675-2ETH (PLC processors should be running firmware version 3.2 or later – use an SD card initially if the PLC has been shipped with v3.0 firmware and then the ‘Update Firmware’ facility within Automation Builder to further update the processor as required).
- The PLC also requires a CM579-ECAT communication module (again, once the PLC is running v3.2 or later this can be updated through Automation Builder). 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 PM5670 PLC with CM579-ETHCAT coupler
- Ethernet cable to connect the PLC EtherCAT coupler to the drive

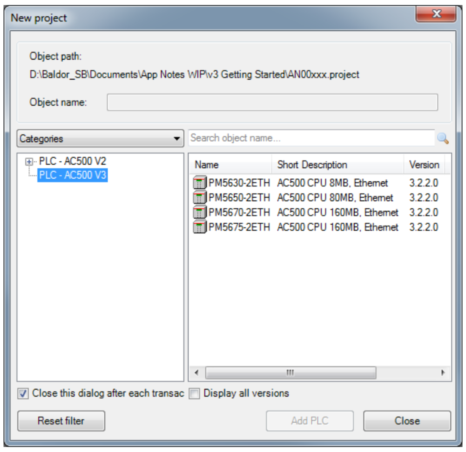
It is assumed the reader has a basic working knowledge of Automation Builder, CoDeSys v3 and the AC500 PLC and that the connected drive has been commissioned / fine-tuned as necessary and is ready to be controlled over EtherCAT.

Drive set-up

This application note assumes that you have already commissioned the drive. i.e. 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. 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.


New project

Object path:
D:\Baldor_SB\Documents\App Notes WIP\v3 Getting Started\AN00xxx.project

Object name:

Categories

- PLC - AC500 V2
- PLC - AC500 V3

Name	Short Description	Version
PM5630-2ETH	AC500 CPU 8MB, Ethernet	3.2.2.0
PM5650-2ETH	AC500 CPU 80MB, Ethernet	3.2.2.0
PM5670-2ETH	AC500 CPU 160MB, Ethernet	3.2.2.0
PM5675-2ETH	AC500 CPU 160MB, Ethernet	3.2.2.0

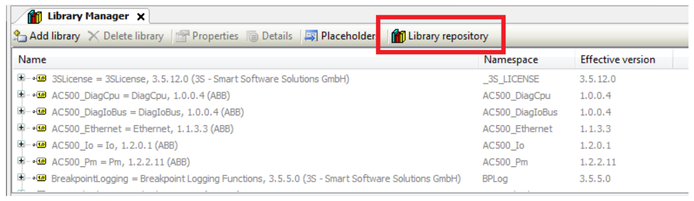
Select 'PLC – AC500 v3' and then select your processor model from the right-hand pane. Click on 'Add PLC' when done. Automation Builder will create a device tree and will give the PLC a name based on the processor type selected. If you like you can click on the device name in the tree to overwrite this and call the device anything you like.

Next, we will add the motion library files to the project. These files are currently unlicensed (usually the library is distributed as a setup/exe and needs a license to be transferred to the PLC) …

- ABB_MathFunctions_AC500.compliled-library
- ABB_MotionControl_AC500_506.compiled-library
- ECAT_AC500_APPL_V02.library

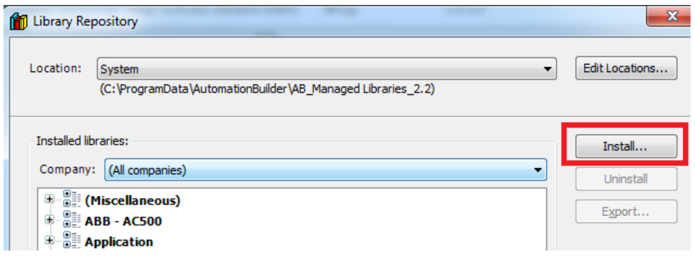
These files are included with this document, copy them to somewhere you can find easily.

Now we need to register these libraries with Automation Builder (this only needs to be done once, Automation Builder will know these libraries exist for future projects). Double click 'Library Manager' in your project tree and in the right-hand pane click on 'Library repository'…



Name	Namespace	Effective version
3SLicense = 3SLicense, 3.5.12.0 (3S - Smart Software Solutions GmbH)	_3S_LICENSE	3.5.12.0
AC500_DiagCpu = DiagCpu, 1.0.0.4 (ABB)	AC500_DiagCpu	1.0.0.4
AC500_DiagIoBus = DiagIoBus, 1.0.0.4 (ABB)	AC500_DiagIoBus	1.0.0.4
AC500_Ethernet = Ethernet, 1.1.3.3 (ABB)	AC500_Ethernet	1.1.3.3
AC500_Io = Io, 1.2.0.1 (ABB)	AC500_Io	1.2.0.1
AC500_Pm = Pm, 1.2.2.11 (ABB)	AC500_Pm	1.2.2.11
BreakpointLogging = Breakpoint Logging Functions, 3.5.5.0 (3S - Smart Software Solutions GmbH)	BPLog	3.5.5.0

When the Library repository dialog opens click on 'Install…'


Library Repository

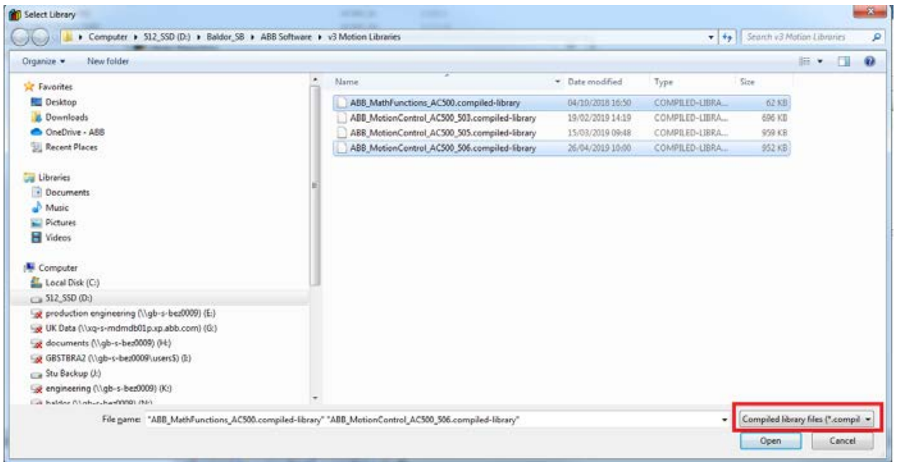
Location: System
(C:\ProgramData\AutomationBuilder\AB_Managed Libraries_2.2)

Installed libraries:

Company: (All companies)

- (Miscellaneous)
- ABB - AC500
- Application

Set the file type selection to 'Compiled library files' and navigate to wherever you stored the library files earlier. You can use the CTRL key to select multiple files…click 'Open' when you've selected the two compiled libraries…


Select Library

Computer ▸ 512_SSD (D:) ▸ Baldor_SB ▸ ABB Software ▸ v3 Motion Libraries

Search v3 Motion Libraries

Organize ▾ New folder

Favorites
Desktop
Downloads
OneDrive - ABB
Recent Places

Libraries
Documents
Music
Pictures
Videos

Computer
Local Disk (C:)
512_SSD (D:)
production engineering (\\gb-s-bez0009) (E:)
UK Data (\\xq-s-mdmdb01p.xp.abb.com) (G:)
documents (\\gb-s-bez0009) (H:)
GBSTBRA2 (\\gb-s-bez0009\users$) (I:)
Stu Backup (J:)
engineering (\\gb-s-bez0009) (K:)

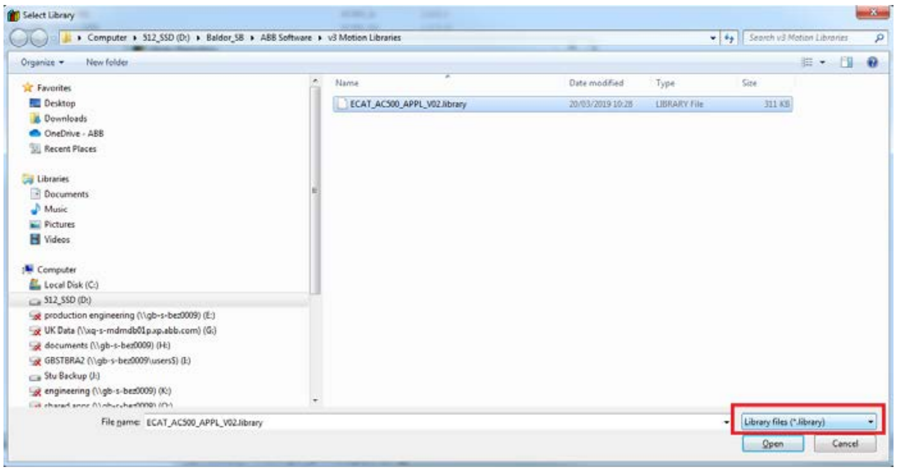
Name	Date modified	Type	Size
ABB_MathFunctions_AC500.compiled-library	04/10/2018 16:50	COMPILED-LIBRA…	62 KB
ABB_MotionControl_AC500_503.compiled-library	19/02/2019 14:19	COMPILED-LIBRA…	696 KB
ABB_MotionControl_AC500_505.compiled-library	15/03/2019 09:48	COMPILED-LIBRA…	959 KB
ABB_MotionControl_AC500_506.compiled-library	26/04/2019 10:00	COMPILED-LIBRA…	952 KB

File name: "ABB_MathFunctions_AC500.compiled-library" "ABB_MotionControl_AC500_506.compiled-library"

Compiled library files (*.compil

Open Cancel

Repeat this process, but this time set the file type selection to 'Library files' and this time select the EtherCAT application library…


Select Library

Computer ▸ 512_SSD (D:) ▸ Baldor_SB ▸ ABB Software ▸ v3 Motion Libraries

Search v3 Motion Libraries

Organize ▾ New folder

Favorites
Desktop
Downloads
OneDrive - ABB
Recent Places

Libraries
Documents
Music
Pictures
Videos

Computer
Local Disk (C:)
512_SSD (D:)
production engineering (\\gb-s-bez0009) (E:)
UK Data (\\xq-s-mdmdb01p.xp.abb.com) (G:)
documents (\\gb-s-bez0009) (H:)
GBSTBRA2 (\\gb-s-bez0009\users$) (I:)
Stu Backup (J:)
engineering (\\gb-s-bez0009) (K:)

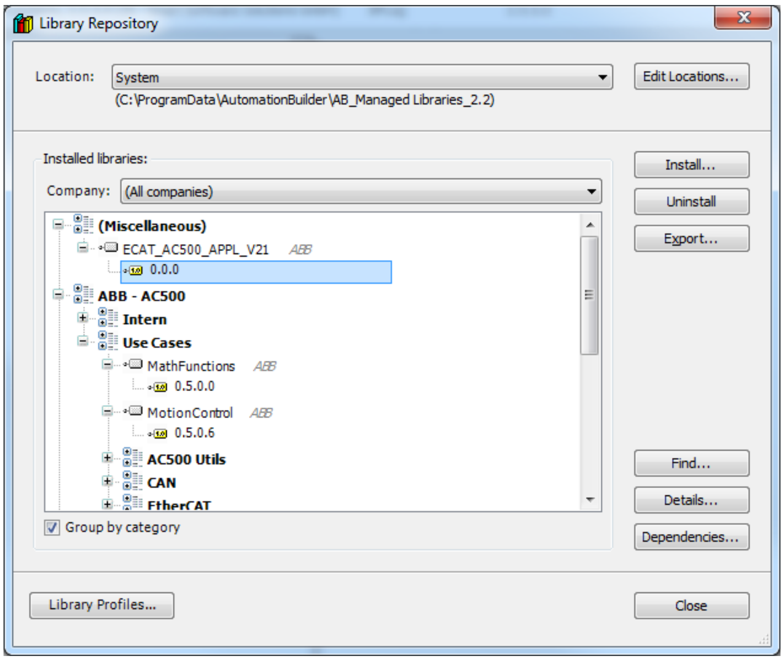
Name	Date modified	Type	Size
ECAT_AC500_APPL_V02.library	20/03/2019 10:28	LIBRARY File	311 KB

File name: ECAT_AC500_APPL_V02.library

Library files (*.library)

Open Cancel

When complete Automation Builder should show that the EtherCAT application library has been registered under the '(Miscellaneous)' section of the repository and the MotionControl and MathFunctions libraries have been registered under 'ABB – AC500 > Use Cases'…


Library Repository

Location: System
(C:\ProgramData\AutomationBuilder\AB_Managed Libraries_2.2)

Edit Locations…

Installed libraries:

Company: (All companies)

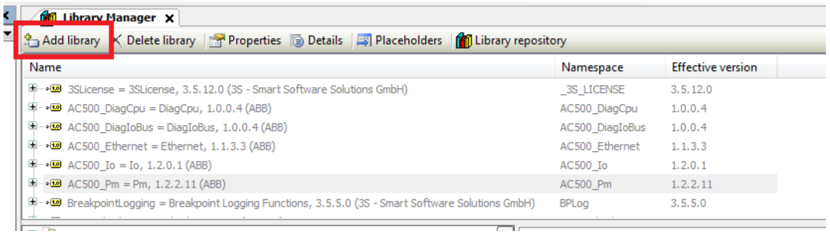
```
(Miscellaneous)
  ECAT_AC500_APPL_V21    ABB
    0.0.0
ABB - AC500
  Intern
  Use Cases
    MathFunctions    ABB
      0.5.0.0
    MotionControl    ABB
      0.5.0.6
    AC500 Utils
    CAN
    EtherCAT
```

☑ Group by category

Install… Uninstall Export… Find… Details… Dependencies…

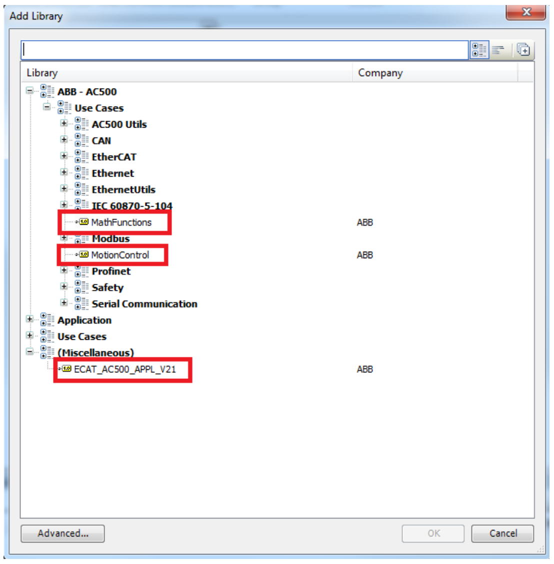
Library Profiles… Close

Now the libraries are in the repository we can close this dialog (remember, you don't need to repeat this when starting new projects).
Next, we need to add these libraries to our project, so this time in the 'Library Manager' window click the 'Add library' button…



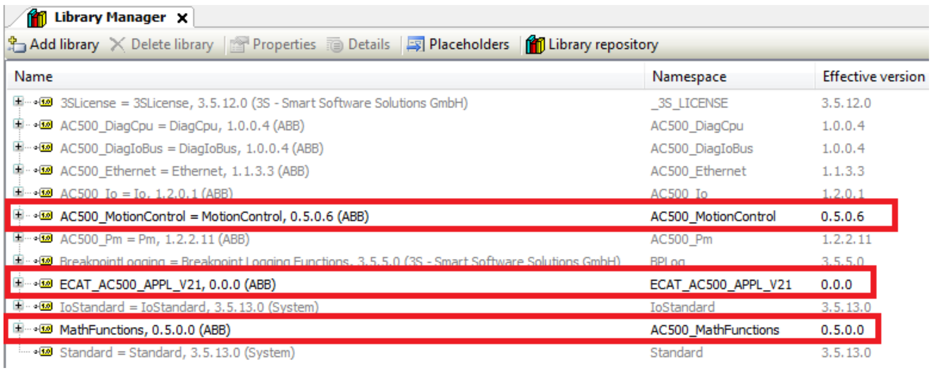
Name	Namespace	Effective version
3SLicense = 3SLicense, 3.5.12.0 (3S - Smart Software Solutions GmbH)	_3S_LICENSE	3.5.12.0
AC500_DiagCpu = DiagCpu, 1.0.0.4 (ABB)	AC500_DiagCpu	1.0.0.4
AC500_DiagIoBus = DiagIoBus, 1.0.0.4 (ABB)	AC500_DiagIoBus	1.0.0.4
AC500_Ethernet = Ethernet, 1.1.3.3 (ABB)	AC500_Ethernet	1.1.3.3
AC500_Io = Io, 1.2.0.1 (ABB)	AC500_Io	1.2.0.1
AC500_Pm = Pm, 1.2.2.11 (ABB)	AC500_Pm	1.2.2.11
BreakpointLogging = Breakpoint Logging Functions, 3.5.5.0 (3S - Smart Software Solutions GmbH)	BPLog	3.5.5.0

If you know the name of the library to install you can type some text to search for it (e.g. “motion” would find the MotionControl library). Alternatively, we just saw that the MotionControl and MathFunctions libraries were registered at ‘ABB – AC500 > Use Cases’ and the EtherCAT application library was registered at ‘(Miscellaneous)’ so we can just open these locations to find the libraries…



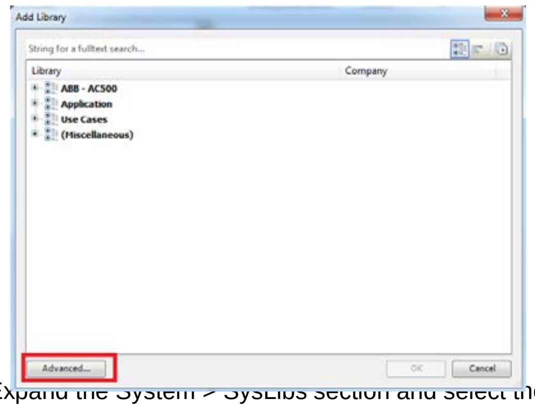
Library	Company
ABB - AC500	
Use Cases	
AC500 Utils	
CAN	
EtherCAT	
Ethernet	
EthernetUtils	
IEC 60870-5-104	
MathFunctions	ABB
Modbus	
MotionControl	ABB
Profinet	
Safety	
Serial Communication	
Application	
Use Cases	
(Miscellaneous)	
ECAT_AC500_APPL_V21	ABB

Select each library in turn and click the ‘OK’ button. As each is selected, they will appear (in bold) in the top pane of the Library Manager…

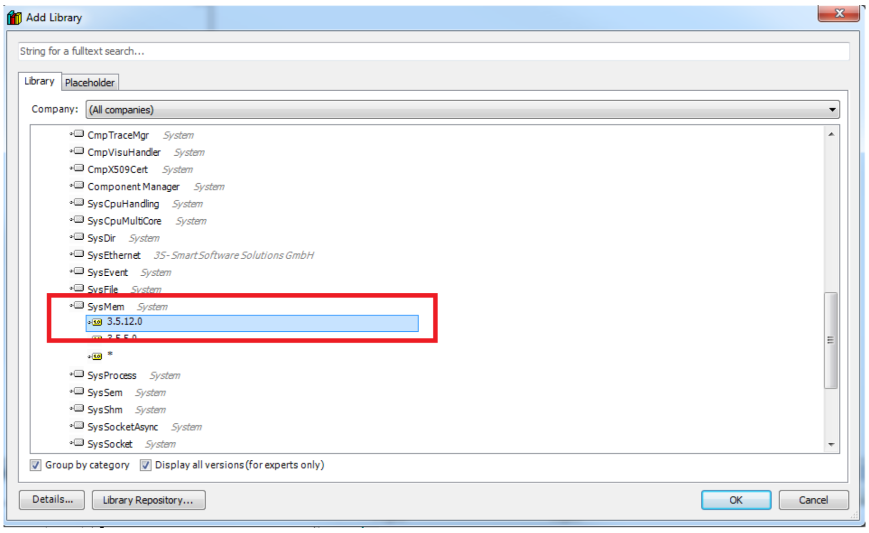


Name	Namespace	Effective version
3SLicense = 3SLicense, 3.5.12.0 (3S - Smart Software Solutions GmbH)	_3S_LICENSE	3.5.12.0
AC500_DiagCpu = DiagCpu, 1.0.0.4 (ABB)	AC500_DiagCpu	1.0.0.4
AC500_DiagIoBus = DiagIoBus, 1.0.0.4 (ABB)	AC500_DiagIoBus	1.0.0.4
AC500_Ethernet = Ethernet, 1.1.3.3 (ABB)	AC500_Ethernet	1.1.3.3
AC500_Io = Io, 1.2.0.1 (ABB)	AC500_Io	1.2.0.1
AC500_MotionControl = MotionControl, 0.5.0.6 (ABB)	AC500_MotionControl	0.5.0.6
AC500_Pm = Pm, 1.2.2.11 (ABB)	AC500_Pm	1.2.2.11
BreakpointLogging = Breakpoint Logging Functions, 3.5.5.0 (3S - Smart Software Solutions GmbH)	BPLog	3.5.5.0
ECAT_AC500_APPL_V21, 0.0.0 (ABB)	ECAT_AC500_APPL_V21	0.0.0
IoStandard = IoStandard, 3.5.13.0 (System)	IoStandard	3.5.13.0
MathFunctions, 0.5.0.0 (ABB)	AC500_MathFunctions	0.5.0.0
Standard = Standard, 3.5.13.0 (System)	Standard	3.5.13.0

Some of our EtherCAT related function blocks we will add later will make use of the system function ‘SysMemSwap’ so we also need to ensure the library that includes this is included in our project. Click on the ‘Add library’ button again and this time click the ‘Advanced’ button on the resulting dialog…



Expand the System > SysLibs section and select the latest available version of ‘SysMem’ – click OK to include this in the project…

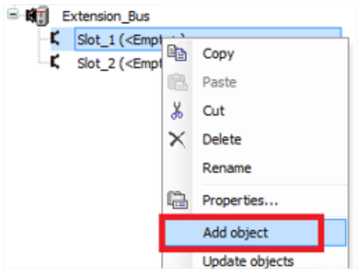


Company: (All companies)

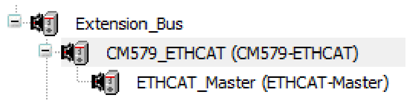
- CmpTraceMgr *System*
- CmpVisuHandler *System*
- CmpX509Cert *System*
- Component Manager *System*
- SysCpuHandling *System*
- SysCpuMultiCore *System*
- SysDir *System*
- SysEthernet *3S-Smart Software Solutions GmbH*
- SysEvent *System*
- SysFile *System*
- SysMem *System*
  - 3.5.12.0
  - 3.5.5.0
  - *
- SysProcess *System*
- SysSem *System*
- SysShm *System*
- SysSocketAsync *System*
- SysSocket *System*

[✓] Group by category [✓] Display all versions (for experts only)

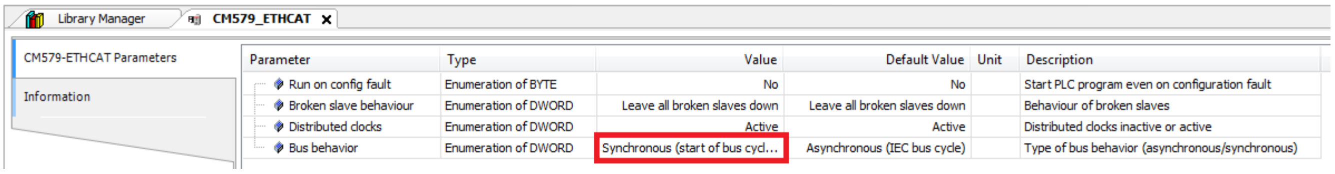
Next, we can add the EtherCAT coupler to our project tree. Right-click the Slot_1 position on the Extension Bus and select 'Add object'…



Select 'CM579-ETHCAT' from the list of available options. This module will now appear in the project tree…



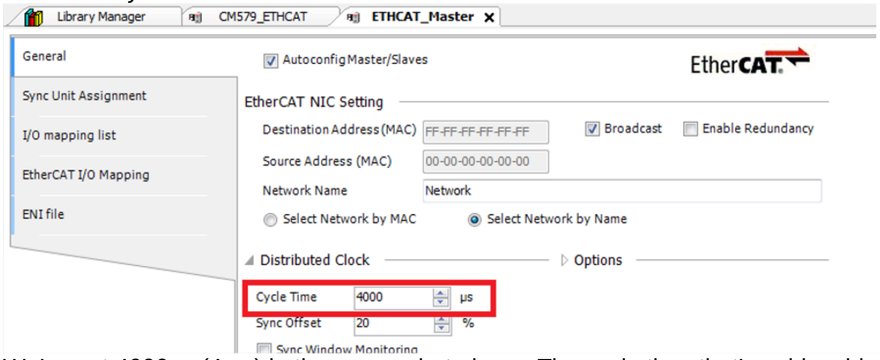
Double-click the 'CM579_ETHCAT' entry and select 'CM579-ETHCAT Parameters' in the right-hand pane. Set the 'Bus behavior' parameter to 'Synchronous (start of bus cycle)' as shown below…



Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
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
Bus behavior	Enumeration of DWORD	Synchronous (start of bus cycl…	Asynchronous (IEC bus cycle)		Type of bus behavior (asynchronous/synchronous)

It should also be possible to use 'end of bus cycle' but there is a bug in v3 at present that means 'start of bus cycle' is the only mode that works.

Now double-click the 'ETHCAT_Master' entry in the project tree and on the General page in the right-hand pane set the required EtherCAT cycle time…



[✓] Autoconfig Master/Slaves

EtherCAT NIC Setting

Destination Address (MAC)	FF-FF-FF-FF-FF-FF	[✓] Broadcast	[ ] Enable Redundancy
Source Address (MAC)	00-00-00-00-00-00		
Network Name	Network		

Select Network by MAC [✓ Select Network by Name]

Distributed Clock Options

Cycle Time	4000	µs
Sync Offset	20	%

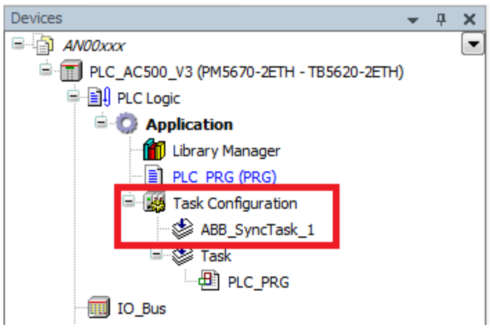
[ ] Sync Window Monitoring

We've set 4000us (4ms) in the screenshot above. The cycle time that's achievable depends on the processor specification, the number of axes and whether the application needs to use Touchprobe or any other function blocks and logic within the tasks synchronised with the EtherCAT task. Performance should be improved for 2020, for now it is sensible to use processor type: *PM5650* as a minimum level and the benchmarks for this or the higher-level processors can be summarized as…

Cycle Time	No of axes (no touchprobe)	No of axes (with touchprobe)
2	8	2
4	30	6

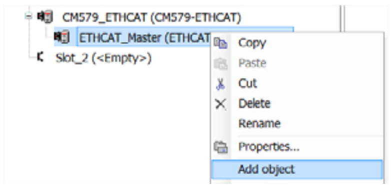
** Note **
Operation faster than 2ms with v3 PLCs is currently not possible.

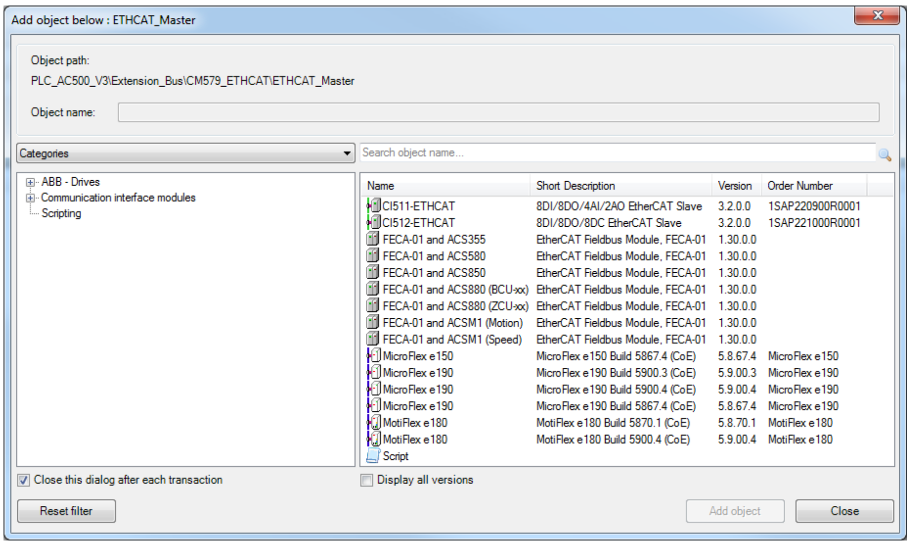
Now we've added the EtherCAT master and setup the bus synchronisation behaviour we can build the project – this will then automatically create the task that will be associated with the EtherCAT data transmission. From the top ribbon select *'Build > Build'* from the top menu in Automation Builder.
Automation Builder will create a new task named 'ABB_SyncTask_1' automatically and show this under the 'Task Configuration' entry in the project tree…



You can rename this task if you like (e.g. Realtime). Later we will add a program call to this task (i.e. call the code that must execute every EtherCAT cycle).

Now we can add a drive to the project tree (either a MicroFlex e190 or a MotiFlex e180). We will use a MicroFlex e190 for this example. Right-click the 'ETHCAT_Master' entry and select 'Add object'…




Add object below : ETHCAT_Master
Categories
Search object name...
ABB - Drives
Communication interface modules
Scripting
Close this dialog after each transaction
Display all versions
Reset filter
Add object
Close

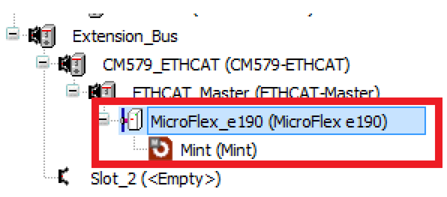
Object path:
PLC_AC500_V3\Extension_Bus\CM579_ETHCAT\ETHCAT_Master
Object name:

Name	Short Description	Version	Order Number
CI511-ETHCAT	8DI/8DO/4AI/2AO EtherCAT Slave	3.2.0.0	1SAP220900R0001
CI512-ETHCAT	8DI/8DO/8DC EtherCAT Slave	3.2.0.0	1SAP221000R0001
FECA-01 and ACS355	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
FECA-01 and ACS580	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
FECA-01 and ACS850	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
FECA-01 and ACS880 (BCU-xx)	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
FECA-01 and ACS880 (ZCU-xx)	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
FECA-01 and ACSM1 (Motion)	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
FECA-01 and ACSM1 (Speed)	EtherCAT Fieldbus Module, FECA-01	1.30.0.0	
MicroFlex e150	MicroFlex e150 Build 5867.4 (CoE)	5.8.67.4	MicroFlex e150
MicroFlex e190	MicroFlex e190 Build 5900.3 (CoE)	5.9.00.3	MicroFlex e190
MicroFlex e190	MicroFlex e190 Build 5900.4 (CoE)	5.9.00.4	MicroFlex e190
MicroFlex e190	MicroFlex e190 Build 5867.4 (CoE)	5.8.67.4	MicroFlex e190
MotiFlex e180	MotiFlex e180 Build 5870.1 (CoE)	5.8.70.1	MotiFlex e180
MotiFlex e180	MotiFlex e180 Build 5900.4 (CoE)	5.9.00.4	MotiFlex e180
Script			

Automation Builder will show a list of all EtherCAT devices that are currently in its 'Device repository'. You should select the device that matches the drive type and *firmware* you are using. These files contain lists of EtherCAT objects that the drive supports, so as new firmware is released the device file may contain additional entries to describe new drive parameters/features – it's therefore important to use the correct file.

** Note **
There are devices relating to specific firmware versions of the drives (E.g. Version 5.9.00.4 = 5900.4 in Mint). If the file that matches the firmware in use on your drive is not present, you should close this window and use 'Tools>Device repository' to register the required file(s). EtherCAT ESI/XML files can be downloaded from the ABB motion website or retrieved from the drive directly via its webpage. See AN00205 for further details if needed.

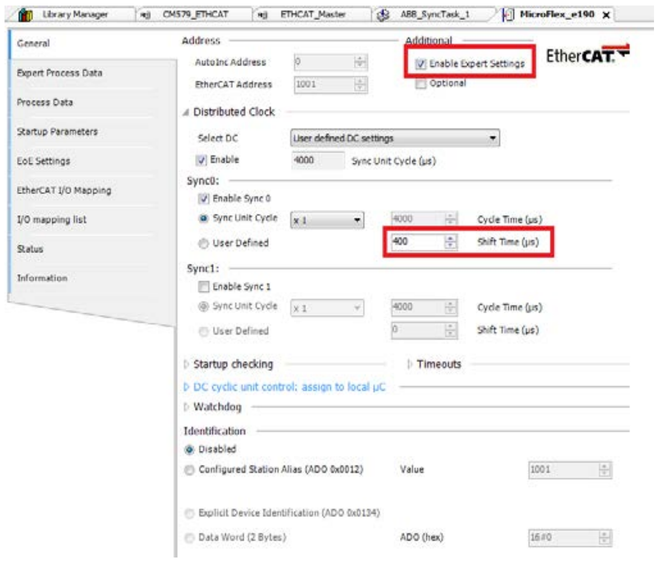
In our case we're using a MicroFlex e190 drive running firmware version 5900.4 so we'll select this entry and click 'Add object'. The drive now appears in the project tree…



```
Extension_Bus
  CM579_ETHCAT (CM579-ETHCAT)
    ETHCAT_Master (ETHCAT-Master)
      MicroFlex_e190 (MicroFlex e190)
        Mint (Mint)
  Slot_2 (<Empty>)
```

** Note **
EtherCAT uses the physical topology to identify each axis/drive, so the order the drives are added to the tree must match the physical order in which the EtherCAT connections are made.

Double-click the drive entry in the tree to access the parameters for the drive. We should select 'Enable Expert Settings' and set the EtherCAT shift time to 10% of the EtherCAT cycle time as shown below…



Address
AutoInc Address: 0
EtherCAT Address: 1001

Additional
[✓] Enable Expert Settings
[ ] Optional

Distributed Clock
Select DC: User defined DC settings
[✓] Enable 4000 Sync Unit Cycle (µs)

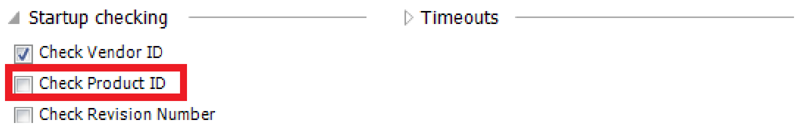
Sync0:
[✓] Enable Sync 0
(●) Sync Unit Cycle x 1 — 4000 Cycle Time (µs)
( ) User Defined — 400 Shift Time (µs)

Sync1:
[ ] Enable Sync 1
(●) Sync Unit Cycle x 1 — 4000 Cycle Time (µs)
( ) User Defined — 0 Shift Time (µs)

Startup checking
Timeouts
DC cyclic unit control: assign to local µC
Watchdog

Identification
(●) Disabled
( ) Configured Station Alias (ADO 0x0012) Value 1001
( ) Explicit Device Identification (ADO 0x0134)
( ) Data Word (2 Bytes) ADO (hex) 16#0

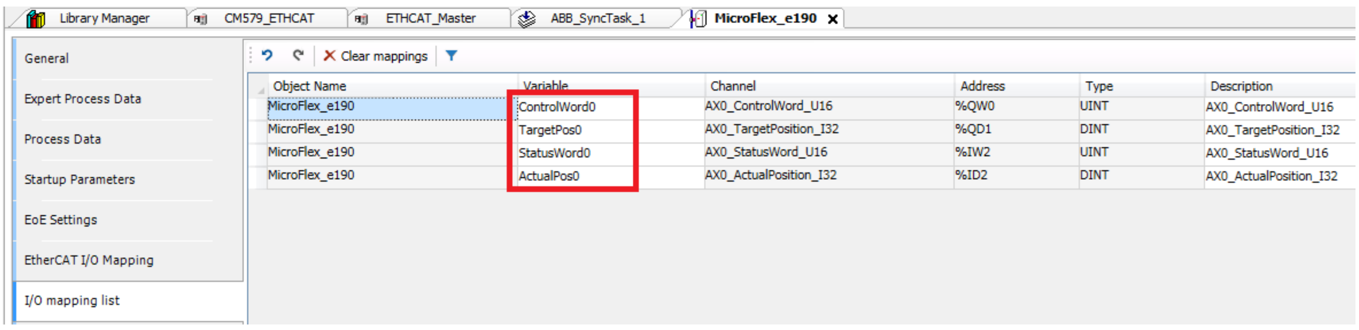
** Note **
The 'Startup checking' section of this pane can be useful if you need to run an application defined to use drives of a type you don't have. For example, imagine a project includes an e190 drive but you only have an e180 drive available for testing. Expand 'Startup checking' and untick the 'Check Product ID' option…



Startup checking
Timeouts
[✓] Check Vendor ID
[ ] Check Product ID
[ ] Check Revision Number

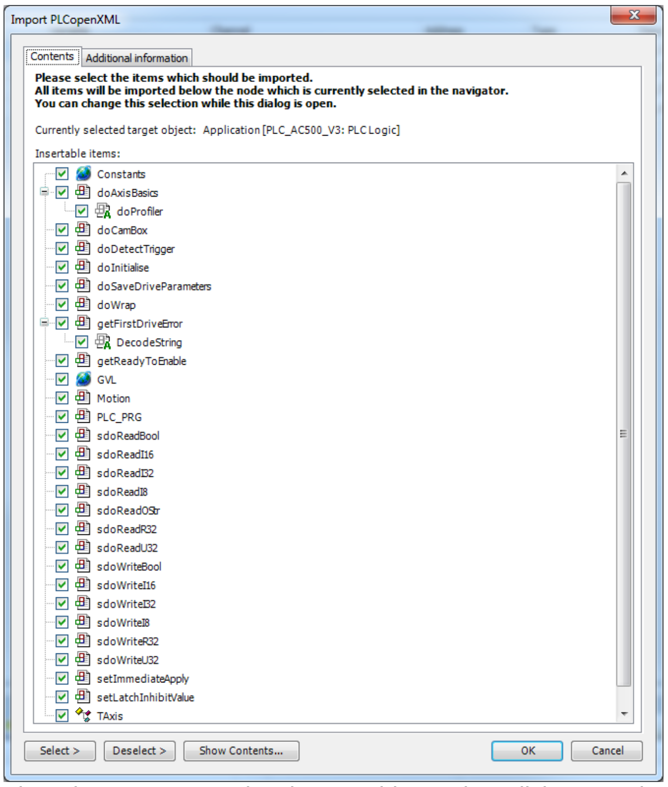
The PLC won't check that the drive is of the correct type and because e180 and e190 drives operate identically you can now use either drive type regardless of what's set in the project tree.

After adding the drive to the tree Automation Builder will automatically add the default PDO mappings needed to control the drive via Cyclic Synchronous Position mode of operation. We need to give "variable names" to these PDO mappings so open the I/O mapping list in the right-hand pane and give the defined mappings sensible names as shown below…



Object Name	Variable	Channel	Address	Type	Description
MicroFlex_e190	ControlWord0	AX0_ControlWord_U16	%QW0	UINT	AX0_ControlWord_U16
MicroFlex_e190	TargetPos0	AX0_TargetPosition_I32	%QD1	DINT	AX0_TargetPosition_I32
MicroFlex_e190	StatusWord0	AX0_StatusWord_U16	%IW2	UINT	AX0_StatusWord_U16
MicroFlex_e190	ActualPos0	AX0_ActualPosition_I32	%ID2	DINT	AX0_ActualPosition_I32

If there are multiple axes in the application repeat these steps to add further drives to the tree and allocate *unique* variable names to their PDO mappings too.
We are now ready to import our v3 motion template which will automatically create data structures and template code for our motion application. Select the 'Application' icon in the project tree. Then from the top menus select 'Project > Import > PLCopen XML…' and navigate to the *'v3 motion template.xml'* file included with this document (wherever you have saved it) and click 'Open'. Automation Builder will show you everything that can now be imported…


Import PLCopenXML

Contents | Additional information

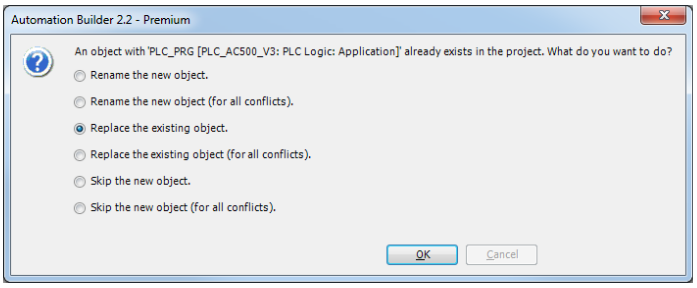
Please select the items which should be imported.
All items will be imported below the node which is currently selected in the navigator.
You can change this selection while this dialog is open.

Currently selected target object: Application [PLC_AC500_V3: PLC Logic]

Insertable items:

- ☑ Constants
- ☑ doAxisBasics
  - ☑ doProfiler
- ☑ doCamBox
- ☑ doDetectTrigger
- ☑ doInitialise
- ☑ doSaveDriveParameters
- ☑ doWrap
- ☑ getFirstDriveError
  - ☑ DecodeString
- ☑ getReadyToEnable
- ☑ GVL
- ☑ Motion
- ☑ PLC_PRG
- ☑ sdoReadBool
- ☑ sdoReadI16
- ☑ sdoReadI32
- ☑ sdoReadI8
- ☑ sdoReadOStr
- ☑ sdoReadR32
- ☑ sdoReadU32
- ☑ sdoWriteBool
- ☑ sdoWriteI16
- ☑ sdoWriteI32
- ☑ sdoWriteI8
- ☑ sdoWriteR32
- ☑ sdoWriteU32
- ☑ setImmediateApply
- ☑ setLatchInhibitValue
- ☑ TAxis

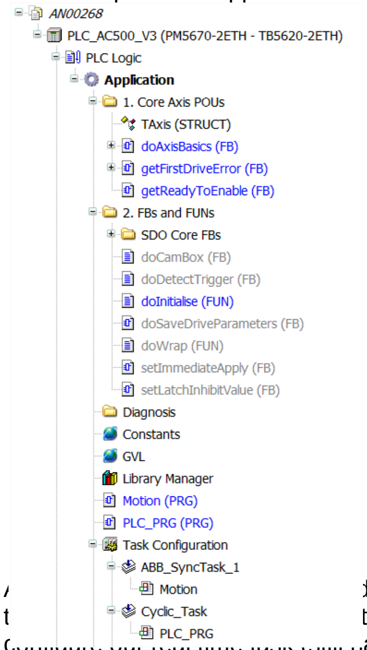
There is no reason to deselect anything so just click ‘*OK’* to import all files. Automation Builder will detect the existence of some POUs already so select ‘Replace the existing object’ whenever you are prompted to make a choice…


Automation Builder 2.2 - Premium

An object with 'PLC_PRG [PLC_AC500_V3: PLC Logic: Application]' already exists in the project. What do you want to do?

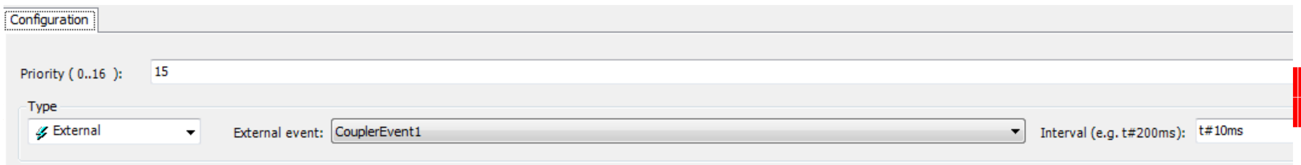
- ○ Rename the new object.
- ○ Rename the new object (for all conflicts).
- ● Replace the existing object.
- ○ Replace the existing object (for all conflicts).
- ○ Skip the new object.
- ○ Skip the new object (for all conflicts).

Once complete the Application section of your project should look like this…


```
AN00268
  PLC_AC500_V3 (PM5670-2ETH - TB5620-2ETH)
    PLC Logic
      Application
        1. Core Axis POUs
          TAxis (STRUCT)
          doAxisBasics (FB)
          getFirstDriveError (FB)
          getReadyToEnable (FB)
        2. FBs and FUNs
          SDO Core FBs
          doCamBox (FB)
          doDetectTrigger (FB)
          doInitialise (FUN)
          doSaveDriveParameters (FB)
          doWrap (FUN)
          setImmediateApply (FB)
          setLatchInhibitValue (FB)
        Diagnosis
        Constants
        GVL
        Library Manager
        Motion (PRG)
        PLC_PRG (PRG)
        Task Configuration
          ABB_SyncTask_1
            Motion
          Cyclic_Task
            PLC_PRG
```

/ [illegible] d ‘Motion’ is included which should contain all code that needs to execute in synchronism with t [illegible] that processes the axis profiling). We will look at how this works later but for now we need to configure our real time task (still named ABB_SyncTask_1 in our example) to call this program.

Double-click your real time task and add the call to the Motion POU…


Configuration

Priority (0..16): 15

Type

External | External event: CouplerEvent1 | Interval (e.g. t#200ms): t#10ms

Now let’s look at the various elements that have been imported…

Core axis POUs

A folder is imported that contains the following…

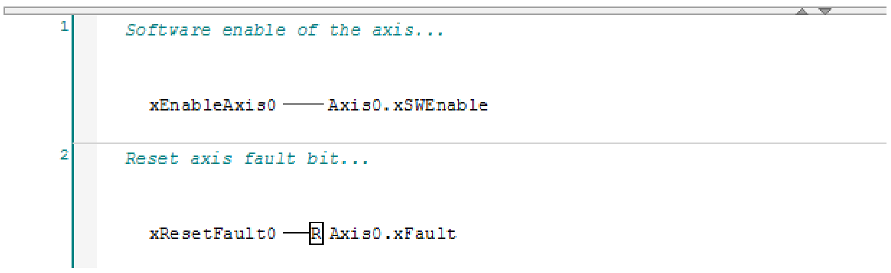
TAxis (Struct) – this is a data structure that effectively creates a “class” for an axis (i.e. it contains key variables relating to the axis, such as the homed status, fault status etc as well as key function blocks for functions that all axes require such as MC_Power for enable/disable, MC_ReadActualPosition to read position and the core parameter, kernel and DS402 blocks needed to control the axis). The template attempts to define everything every application is likely to need “as standard” in this structure (you may add or remove items as you need)

doAxisBasics (FB) – this is a function block that is written to perform all the basic functions every axis is likely to need…

- Operation of the axis profiler and DS402 interface
- Enable / disable of the axis (the xSWEnable, xFault and xDelayedStop member variables of the Axis structure are used in this logic – all active high)
- Reading axis position into member variable Axis[x]. Pos
- Reading axis velocity into member variable Vel
- Reading axis status via member function MC_ReadStatus
- Retrieving error details from the axis and populating member variables Axis[x].dErrCode and Axis[x].strError

** Note **
“member variables are accessed in the form: Axis[x].[member] where ‘x’ is the axis number (you have designated) and ‘member’ is the data you want to interact with.

xFault is *latched* in the event of a drive error. User application code (e.g. in a background cyclic task) can be used to unlatch this variable as needed (a global variable xResetFault0 is provided as the input mechanism which in turn can unlatch the fault variable).
Similarly, a global variable xEnableAxis0 is provided which can be used to control member variable xSWEnable. This is illustrated by the code that is imported by default into the PLC_PRG program which is called cyclically by the default task…



```
1   Software enable of the axis...

    xEnableAxis0 ——— Axis0.xSWEnable

2   Reset axis fault bit...

    xResetFault0 ——[R] Axis0.xFault
```

The remaining function blocks within this folder are those that are invoked by doAxisBasics.

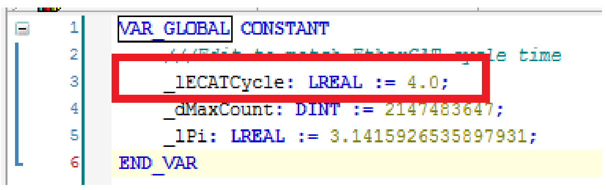
FBs and FUNs

This folder contains a collection of useful function blocks and functions, not all of which will be required by every application but may be very useful. The SDO Core FBs sub-folder contains function blocks to perform SDO read/write operations (e.g. drive parameter/object access) and some of these are inherently called by other elements of the project (e.g. the getFirstDriveError function block needs to use SDO access to retrieve error information).
The function of each entry in the folder is briefly described below…

- doCamBox – a function block that turns on an output of the block when an input value is within a specified range
- doDetectTrigger – a function block that turns on an output when a value passes a particular value (can be used to replicate “trigger on position” type functionality from Mint based controllers)
- doInitialise – this is used on every application. This function must be called by the ‘Prepare to start’ system event and contains code to parameterise each axis in the application (e.g. sets scaling, determines if the axis is non-modulo or modulo). We will detail how to call and edit this routine later
- doSaveDriveParameters – function block used to permanently store drive parameters
- doWrap – function block used to ensure a value remains within a specific range or can be used to detect the distance remaining or shortest path between two values (similar in operation to the Mint WRAP keyword)
- setImmediateApply – function block to control the drive’s immediate apply object (which needs to be turned on in order for EtherCAT object values to be passed across to their Mint side equivalents – e.g. when setting Sentinel values from the PLC)
- setLatchInhibitValue – function block used to configure latch inhibit values on the drive itself so the drive can automatically filter latch events

Constants

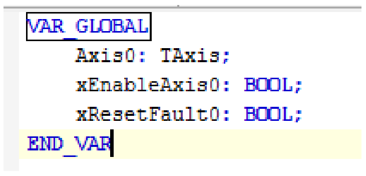
A list of constants most applications are likely to use, including definition of the EtherCAT cycle time. This MUST be edited to suit the value set earlier on the EtherCAT master (e.g. we set 4000us earlier [4ms] so we must edit our constant file to set 4.0 for the cycle time) …



```
VAR_GLOBAL CONSTANT
    //Edit to match EtherCAT cycle time
    _lECATCycle: LREAL := 4.0;
    _dMaxCount: DINT := 2147483647;
    _lPi: LREAL := 3.1415926535897931;
END_VAR
```

GVL

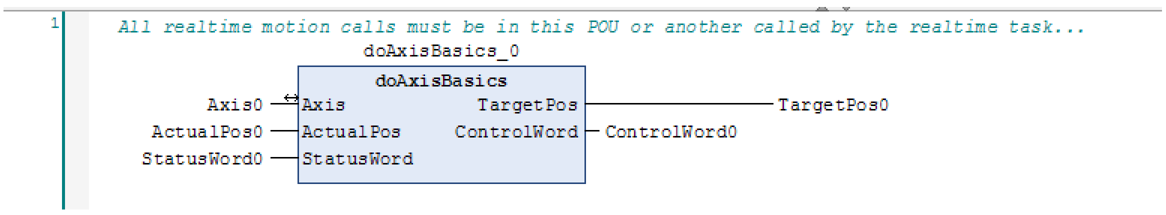
A global variable list that by default defines an Axis of type Taxis as well as the fault reset, and software enable variables we mentioned earlier…



```
VAR_GLOBAL
    Axis0: TAxis;
    xEnableAxis0: BOOL;
    xResetFault0: BOOL;
END_VAR
```

Motion (PRG)

This is our real time program that we’ve configured to execute whenever the EtherCAT coupler starts a cycle. All the core processing for an axis has been encapsulated into a single doAxisBasics function block call, so for our example this program just contains a call relating to our only axis (Axis0 as defined by our Global Variable List) …

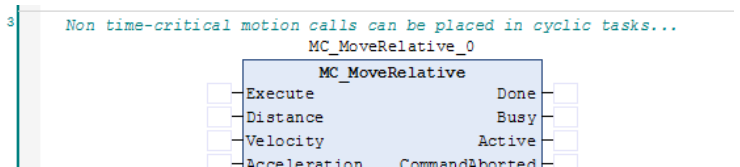


```
All realtime motion calls must be in this POU or another called by the realtime task...
                        doAxisBasics_0
                   ┌──────doAxisBasics──────┐
        Axis0 ↔────┤Axis           TargetPos├──────────── TargetPos0
   ActualPos0 ─────┤ActualPos    ControlWord├─ ControlWord0
  StatusWord0 ─────┤StatusWord              │
                   └────────────────────────┘
```

The doAxisBasics FB takes the Axis (of type TAxis) and the PLC input PDO variables as input parameters. It uses the PLC output PDO variables as FB outputs. If further axes are required add further rungs calling doAxisBasics for each.

PLC_PRG (PRG)

This is our non-time-critical (cyclic) program, called by the cyclic task ‘Cyclic_Task’. In our template we have included an instance of a MC_MoveRelative function block in this program to illustrate that, depending on the application, it’s not strictly necessary to place all motion blocks inside the real time program(s). Only the profiler, Touchprobe and drive based homing function blocks must be within a real time program. Other motion blocks can be placed according to factors such as reaction time, application throughput etc…



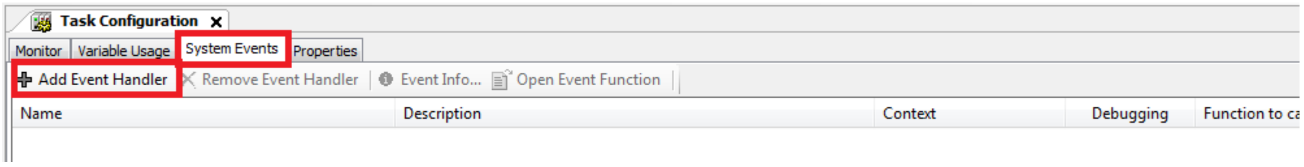
```
Non time-critical motion calls can be placed in cyclic tasks...
                  MC_MoveRelative_0
            ┌──────MC_MoveRelative──────┐
          ──┤Execute                Done├──
          ──┤Distance               Busy├──
          ──┤Velocity             Active├──
          ──┤Acceleration CommandAborted├──
```

** Note **
Motion blocks take an 'Axis reference' as an input parameter. Confusingly these are labelled 'Axis' by the blocks themselves so don't be confused by this and use the Axis (structure) as the input parameter – be sure to use the **AxisRef** member variable of the relevant axis structure (Axis[x].AxisRef). To demonstrate motion a visualization can be added to the project and the built-in MC_MoveRelative visu can be linked to MC_MoveRelative_0.

Task Configuration

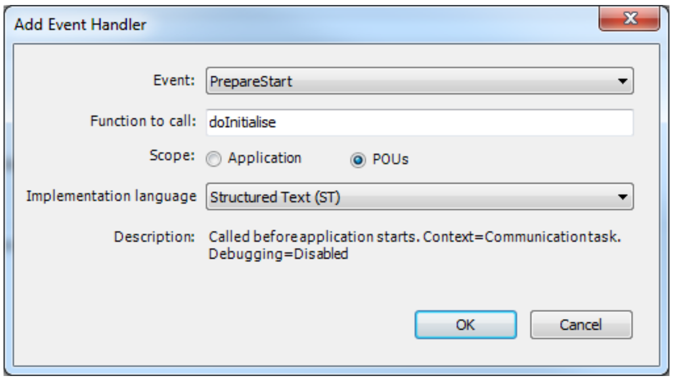
We've already described how the real time task calls Motion (our real time program) and the cyclic task calls PLC_PRG (our background processing task). If we double-click 'Task Configuration' we can see how to configure the application to setup the axis operating parameters…

Select the 'System Events' tab and then click the 'Add Event Handler' button.



The event we need to add is the 'PrepareStart' event so select this from the list of available events.
The function we need to call is named 'doInitialise' so type this name into the 'Function to call' box (this function is written in Structured Text so ensure the 'Implementation language' is set to 'Structured Text (ST)' also. The Scope must be set to POUs.

The resulting dialog should look like this…


Add Event Handler

Event: PrepareStart

Function to call: doInitialise

Scope: Application [✓ POUs]

Implementation language Structured Text (ST)

Description: Called before application starts. Context=Communication task. Debugging=Disabled

Click OK to add this system event.

** Note **
Automation Builder has been seen to delete the existing doInitialise function and create a new blank POU at the application level – if this happens you will need to recreate the contents of the doInitialise POU.

Double-click the doInitialise function block inside the 'FBs and FUNs' folder…

```
// Axis 0 parameters
Axis0.bySlot := 1;       (* On EtherCAT coupler 1 *)
Axis0.dwNode := 1001;    (* First node on the network *)
Axis0.MaxSpeed := 4500;  (* RPM *)
Axis0.Modulo := TRUE;
Axis0.MotorResolution := 131072;  (* Counts per motor rev *)
Axis0.UPRNumerator := 1;    (* 1 motor rev = 1 user unit - i.e. scaled into revs *)
Axis0.UPRDenominator := 1;  (* Numerator / Denom = number of user units in one motor rev *)
Axis0.ModuloVal := LREAL_TO_DINT(_dMaxCount / 2);  (* Modulo set to max value *)
Axis0.Virtual := FALSE;
```

This section should be edited to suit the application…

Axis Structure member	Explanation	What value to use?
bySlot	EtherCAT coupler Slot (slot 1 is the slot nearest the processor)	Set to suit whichever slot the EtherCAT coupler is fitted to.
dwNode	Physical Axis node address for this axis (1001 = 1st drive, 1002 = 2nd drive etc…)	Set according to the physical position on the EtherCAT network
MaxSpeed	Maximum Axis Speed	Set to match the 'Application Max Speed' configured on the drive
Modulo *	Does the axis value "wrap" at a value?	Set to true for a cyclic axis or an axis that runs continually in one direction
MotorResolution	Defines the scaling of the axis "user units".	Set to match the effective resolution of the axis (including pre-scaling). E.g. If its set to encoder resolution the scale will be revolutions.
UPRNumerator	These two values together determine how many user units there are in one revolution of the motor. Overall value = numerator / denominator.	Example sets a user unit of motor revs (i.e. there is 1 /1 unit in one motor rev). Imagine that one motor rev moves the axis 10 and 1/3rd mm. It's possible to set the numerator to 10.3333333 (because it's a LREAL) and the denominator to 1 (because is a DINT) but the PLC application would be more accurate (and avoid position drift over time) by setting the numerator to 31.0 and the denominator to 3
UPRDenominator		
ModuloVal *	If Modulo = TRUE then this value is set to the number of *encoder counts* in one axis cycle	E.g. if the axis was a turntable set to modulo at 360 degrees this value would equate to the number of encoders counts in 360 degrees of travel).
Virtual	Do you have a physical axis to be controlled or will you use a virtual axis?	Set to FALSE for a real EtherCAT axis (the doAxisBasics function block and related functions would need small modifications to suit virtual axes as there are no PDO mappings associated with a virtual axis – in this case it's best to copy doAxisBasics and create a virtual version)

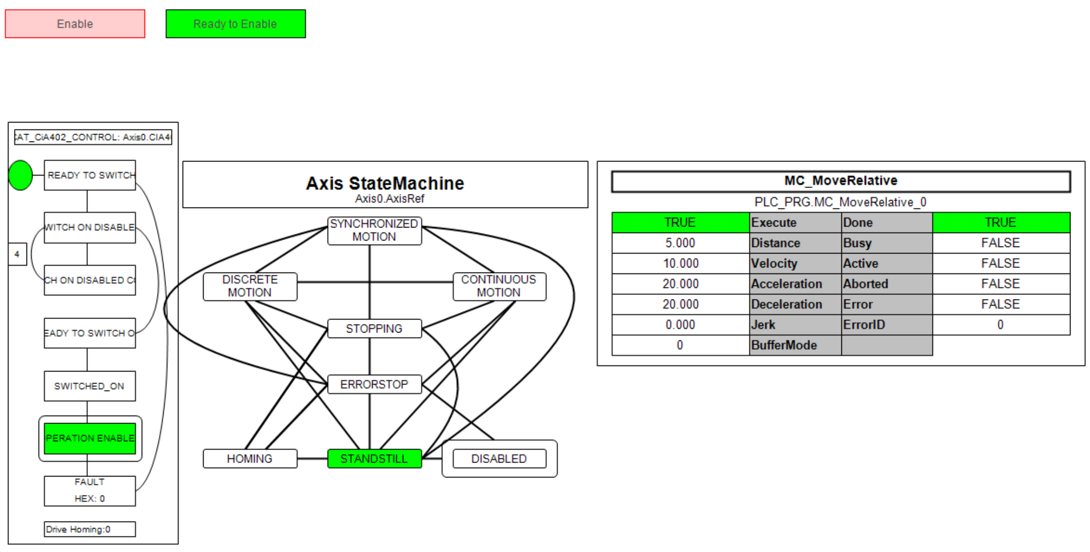
*. For axes that continuously run in one direction (that are also treated as modulo axes because they eventually reach a position wrap) this value is often set to the maximum possible modulo value as shown in the template

At this stage try building the project again (Build > Build) …. if you've followed everything correctly the project should build with no errors (there will be some warnings, mainly because of implicit data type conversion within the motion libraries). If you download and run this program and force xEnableAxis0 true/false you should find that the drive enables/disables (providing it is ready to be enabled – e.g. AC power is applied, there are no drive errors and STO is inactive).

The Automation Builder project included with this document includes a sample visualization that provides the following functions…

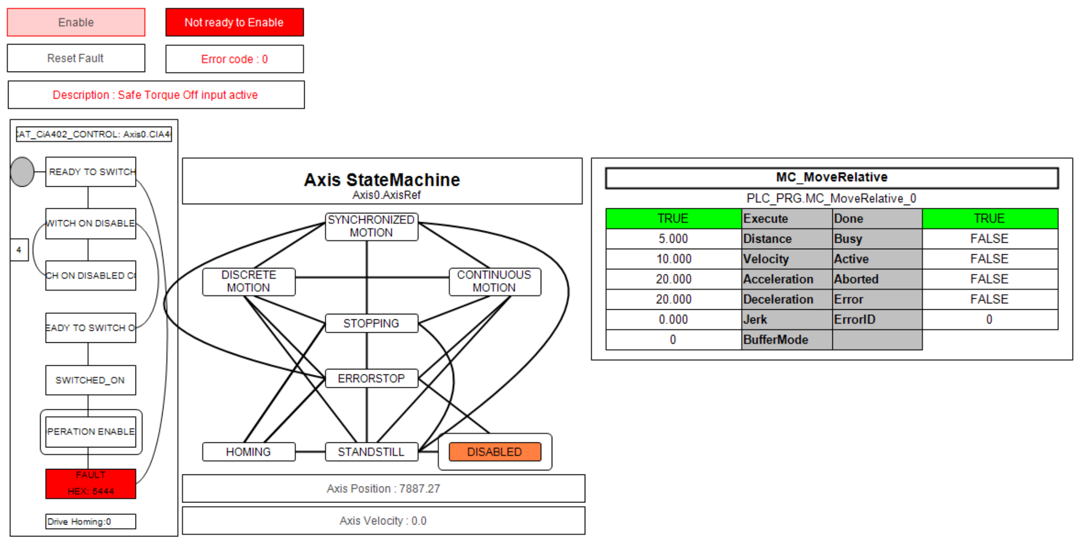
(1) Ready to Enable status
(2) Enable / disable of the axis
(3) Fault reset (button appears when axis is in fault)

(4) MC_MoveRelative visu (enter parameters and set Execute to TRUE to trigger move (linked to MC_MoveRelative_0 in PLC_PRG)
(5) PLCopen state diagram
(6) DS402 axis state diagram
(7) Position and velocity readouts
(8) Display of last error code and description (only visible when axis is in fault state)



MC_MoveRelative
PLC_PRG.MC_MoveRelative_0

TRUE	Execute	Done	TRUE
5.000	Distance	Busy	FALSE
10.000	Velocity	Active	FALSE
20.000	Acceleration	Aborted	FALSE
20.000	Deceleration	Error	FALSE
0.000	Jerk	ErrorID	0
0	BufferMode		



Error code : 0

Description : Safe Torque Off input active

MC_MoveRelative
PLC_PRG.MC_MoveRelative_0

TRUE	Execute	Done	TRUE
5.000	Distance	Busy	FALSE
10.000	Velocity	Active	FALSE
20.000	Acceleration	Aborted	FALSE
20.000	Deceleration	Error	FALSE
0.000	Jerk	ErrorID	0
0	BufferMode		

Axis Position : 7887.27

Axis Velocity : 0.0

Contact Us

For more information please contact your
local ABB representative or one of the following:

www.abb.com/motion
www.abb.com/drives
www.abb.com/drivespartners
www.abb.com/PLC

EtherCAT® is a registered trademark and patented technology, licenced by Beckhoff Automation GmbH, Germany

© Copyright 2019 ABB. All rights reserved.
Specifications subject to change without notice.