ETHERNET/IP SET UP GUIDE

Riccardo Piccinini
Mini Motor S.P.A.

FBS-EIP-EN
Firmware Revision: 5.005

Version 9, 10-2026

Table of Contents

1. DS402 Getting Started 1
- 1.1. What is DS402 1
- 1.2. The state machine 3
- 1.3. Enabling the motor in code 5
- 1.4. Profile Velocity 6
- 1.5. Profile Position 7
- 1.6. Homing 8
- 1.7. From here on 9
- 1.8. EtherNet/IP specifics: fixed assemblies and acyclic messaging 9

2. EtherNet/IP 10
- 2.1. CIP Objects 10
- 2.2. Process Data 14

3. Setting Up the Project 24
- 3.1. Tools 24
- 3.2. Overview 24
- 3.3. Configure Controller Address 24
- 3.4. Configure the Motor Address 26
- 3.5. Install EDS 26
- 3.6. Add DBS Slave to a Project 29
- 3.7. AOI Import 31
- 3.8. Choosing the AOI Family 33
- 3.9. Which Revision This Guide Describes 34

4. DS402 AOI 35
- 4.1. MM_ReadData_dfl 35
- 4.2. MM_SendData_dfl 35
- 4.3. MM_Profile_Homing_dfl 36
- 4.4. MM_Profile_Torque_dfl 36
- 4.5. MM_Profile_Velocity_dfl 37
- 4.6. MM_Profile_Position_dfl 37
- 4.7. MM_Status_dfl 38
- 4.8. MM_Reset_Error_dfl 39

5. Rockwell-Like AOI 40
- 5.1. Common Behaviour 40
- 5.2. Ladder or Structured Text 40
- 5.3. Equivalence with the DS402 AOI 41
- 5.4. MM_MAUI 41
- 5.5. MM_MSO 41
- 5.6. MM_MSF 42
- 5.7. MM_MAFR 42
- 5.8. MM_MAM 42
- 5.9. MM_MAH 43
- 5.10. MM_MAS 44
- 5.11. MM_MAJ 44
- 5.12. MM_MSL 45
- 5.13. MM_MAUO 46

6. CIP Sync AOI 47
- 6.1. How It Works 47
- 6.2. Requirements 47
- 6.3. Setting Up the Project 47

6.4. Data Types 48
6.5. The Blocks 49
6.6. Example Program 53
6.7. Tuning 58
6.8. Diagnostics 59
6.9. Test Setup 59
7. Writing Parameters with Async Messaging 60
8. Micro850 Application Example 62
8.1. Which CPU 62
8.2. Add the DBS as a Generic Device 62
8.3. Process Data Seen From CCW 63
8.4. Structured Text Example 64
8.5. Reading and Writing Parameters from CCW 68
8.6. Commissioning and Diagnostics 69
Appendix A: Updating the Rockwell-Like AOI from Revision 1.0 70

1. DS402 Getting Started

This chapter is a hands-on introduction to controlling a Mini Motor servo drive over a fieldbus using the **CANopen DS402** (CiA 402) device profile. It explains the protocol, walks through the drive state machine and shows, with ready-to-read code, how to enable the motor and command it in *Profile Velocity* and *Profile Position*. Once these concepts are clear, the vendor-specific Application Example that follows will be much easier to read: it is the same logic applied to a concrete PLC.

1.1. What is DS402

CiA 402 (also known as **DS402** / **DSP402**, standardised as *IEC 61800-7-201/301*) is the device profile for drives and motion control. It defines, independently from the underlying fieldbus:

- an **object dictionary**: every quantity exchanged with the drive (commands, references, state, feedback) is an object addressed by a 16-bit hexadecimal index, e.g. `6040h`;
- a **state machine** that governs when power is applied to the motor;
- a set of **modes of operation** (Profile Position, Profile Velocity, Profile Torque, Homing, …);
- the meaning of the two central objects, **Control Word** (`6040h`) and **Status Word** (`6041h`).

The value of the profile is interoperability: the same objects and the same control logic apply on **CANopen**, **EtherCAT**, **EtherNet/IP**, **PROFINET** and **POWERLINK**. Only the transport layer changes; the code you write to talk to the motor stays the same. Modbus is the only exception — it does not use DS402 but a set of dedicated registers.

1.1.1. The master/slave dialogue

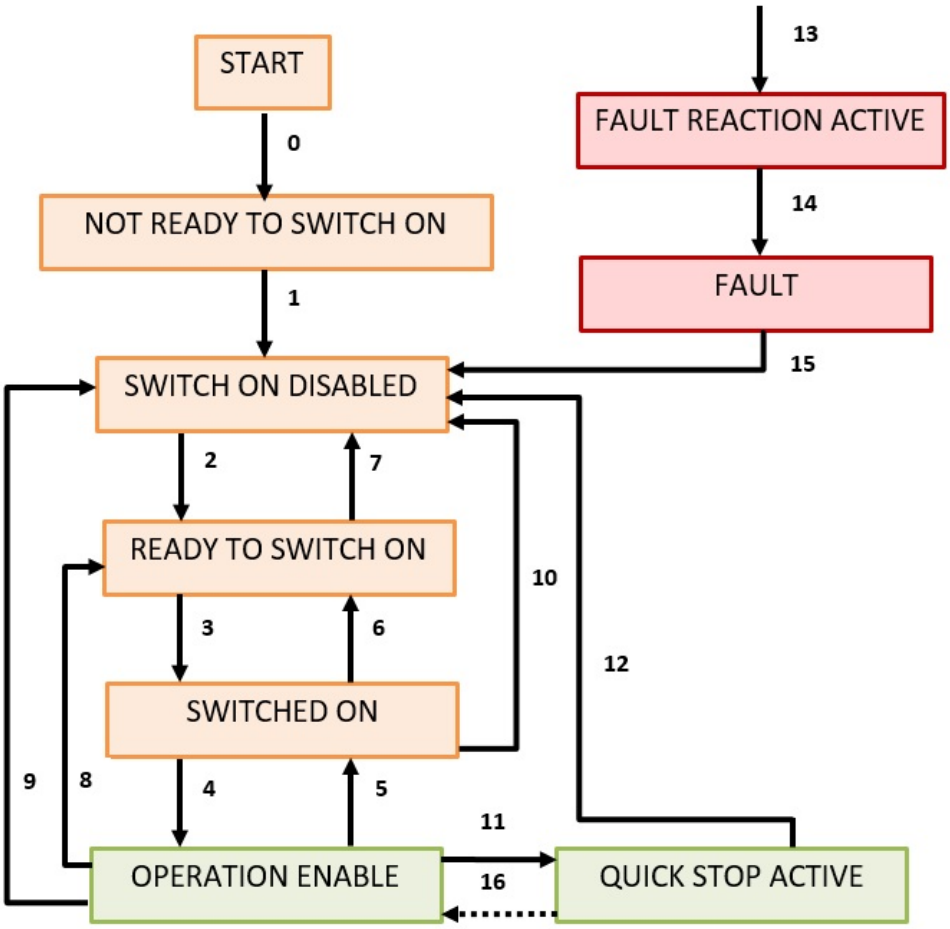
Whatever the mode, the interaction between the master (the controller) and the slave (the servo drive) always follows the same pattern:

1. The master selects the **mode of operation** by writing object *Modes of Operation* (`6060h`); the drive confirms it in *Modes of Operation Display* (`6061h`).
2. The master brings the drive to the *Operation Enabled* state by writing the **Control Word** (`6040h`), following the state machine described below.
3. The master writes the **reference** for the active mode (target velocity `60FFh`, target position `607Ah`, target torque `6071h`, …) and, where required, triggers the motion.
4. The master monitors the drive through the **Status Word** (`6041h`) and the actual-value objects (position `6064h`, velocity `606Ch`, …).

On cyclic fieldbuses these objects are mapped into process data (PDO) and exchanged every cycle; on CANopen they are also reachable via SDO. From the application point of view, you are always reading and writing the same objects.

1.2. The state machine

The motor produces torque only in the **Operation Enabled** state. To get there, the master walks the DS402 finite state machine by writing the Control Word and observing the Status Word.



States in **orange** have power disabled on the motor, states in **green** have power enabled, states in **red** are alarm states.

State	Meaning
Switch On Disabled	Drive ready to be prepared; parameters can be transferred, power can be enabled.
Ready To Switch On	Parameters can be transferred, power can be enabled, the motor cannot run yet.
Switched On	Same as above, one transition away from running.
Operation Enabled	No fault present, motor functions and power enabled: the motor obeys the active mode.
Quick Stop Active	The drive stopped on the emergency ramp; power still enabled, functions disabled.
Fault	A fault is active, the drive is stopped and disabled.

1.2.1. Control Word and Status Word

The transitions are driven by a few bit patterns of the **Control Word** (`6040h`). In practice you only need these commands:

Command	Control Word	Resulting transition
Shutdown	`16#0006`	to *Ready To Switch On*
Switch On	`16#0007`	to *Switched On*
Enable Operation	`16#000F`	to *Operation Enabled* (motor runs)
Disable Voltage	`16#0000`	to *Switch On Disabled*
Quick Stop	`16#0002`	to *Quick Stop Active*
Fault Reset	`16#0080`	clears a fault (acts on the rising edge of bit 7)

The current state is read back from the **Status Word** (`6041h`). By masking the low bits you can tell exactly where the drive is:

Status Word (binary)	State
`xxxx xxxx x1xx 0000`	Switch On Disabled
`xxxx xxxx x01x 0001`	Ready To Switch On
`xxxx xxxx x01x 0011`	Switched On
`xxxx xxxx x01x 0111`	Operation Enabled
`xxxx xxxx x00x 0111`	Quick Stop Active
`xxxx xxxx x0xx 1000`	Fault

Two more Status Word bits are used constantly in the examples below:

- **bit 10 — Target Reached**: set when the drive has reached the commanded position/velocity.
- **bit 12 — Set Point Acknowledge** (Profile Position): the drive acknowledges that it has latched a new position setpoint.

1.3. Enabling the motor in code

The following Structured Text (IEC 61131-3) snippets are vendor-neutral: they use `M1_Cw` for the Control Word written to the motor, `M1_Sw` for the Status Word read from it, `M1_ModeOfOp` / `M1_ModeOfOpDisplay` for objects `6060h` /`6061h`, and the target/actual objects by name. Map these to the tags of your own PLC.

The recommended pattern is a small state machine that reproduces the DS402 transitions. Every cycle it looks at the Status Word and issues the next Control Word until *Operation Enabled* is reached:

```
(* --- Bring the drive to Operation Enabled --- *)
IF (M1_Sw AND 16#004F) = 16#0008 THEN
    (* Fault: request a fault reset on the rising edge of bit 7 *)
    M1_Cw := 16#0080;
ELSIF (M1_Sw AND 16#004F) = 16#0040 THEN
    (* Switch On Disabled -> Ready To Switch On *)
    M1_Cw := 16#0006;
ELSIF (M1_Sw AND 16#006F) = 16#0021 THEN
    (* Ready To Switch On -> Switched On *)
    M1_Cw := 16#0007;
ELSIF (M1_Sw AND 16#006F) = 16#0023 THEN
    (* Switched On -> Operation Enabled *)
    M1_Cw := 16#000F;
END_IF;
```

The masks (`16#004F`, `16#006F`) isolate the state bits of the Status Word so each pattern matches exactly one state; the constant on the right is the state you are leaving. Once the Status Word settles on `···x01x 0111` (Operation Enabled) the motor obeys the active mode.



The `Fault Reset` command (`16#0080`) acts on the **rising edge** of bit 7. Make sure the Control Word returns to a value with bit 7 low before requesting another reset, otherwise the edge is lost.

1.4. Profile Velocity

In **Profile Velocity** (mode `3`) the drive keeps a target speed, reaching it through the configured acceleration/deceleration ramps. The reference is *Target Velocity* (`60FFh`), in rpm.

The logic is:

1. select the mode (`M1_ModeOfOp := 3`) and wait for the drive to confirm it (`M1_ModeOfOpDisplay = 3`);
2. write the target velocity;
3. drive the state machine to Operation Enabled (`16#000F`) to make the motor run.

```
(* --- Profile Velocity --- *)
M1_ModeOfOp := 3;                       (* 6060h: Profile Velocity *)
M1_TargetVelocity := GUI_TargetRpm;     (* 60FFh: rpm, signed = direction *)

IF (M1_ModeOfOpDisplay = 3) THEN
    M1_Cw := 16#000F;                   (* Enable Operation: the motor runs *)
ELSE
    M1_Cw := 16#0006;                   (* mode not confirmed yet: stay ready *)
END_IF;
```

The sign of `M1_TargetVelocity` sets the direction. Writing a new value changes the speed on the fly; there is no start trigger to send, unlike Profile Position. To stop, either write `0` or drop the Control Word back to `16#0006` (Shutdown).

1.5. Profile Position

In **Profile Position** (mode `1`) the drive builds a trapezoidal trajectory from the current position to *Target Position* (`607Ah`), using *Profile Velocity* (`6081h`) and the acceleration/deceleration objects. Unlike Profile Velocity, a target only becomes active when the master toggles the **New Setpoint** bit (bit 4) of the Control Word; the drive confirms it with **Set Point Acknowledge** (bit 12 of the Status Word). This handshake lets you load a new target at any time without the drive picking it up prematurely.

The sequence for a single move is:

1. select the mode (`M1_ModeOfOp := 1`) and wait for confirmation (`M1_ModeOfOpDisplay = 1`);
2. write target position and profile velocity;
3. hold the drive in Operation Enabled (`16#000F`);
4. pulse bit 4 (`16#001F`) to latch the target — the drive raises Status Word bit 12;
5. clear bit 4 (back to `16#000F`) so the handshake can be repeated for the next move.

```
(* --- Profile Position (single move with New Setpoint handshake) --- *)
M1_ModeOfOp := 1;                          (* 6060h: Profile Position *)
M1_TargetPos := GUI_TargetPos;             (* 607Ah *)
M1_ProfileVel := GUI_ProfileRpm;           (* 6081h, rpm *)

IF (M1_ModeOfOpDisplay = 1) THEN
    IF NewSetReq THEN
        (* raise New Setpoint (bit 4) to latch the target *)
        M1_Cw := 16#001F;
        (* the drive acknowledges with Status Word bit 12 (Set Point Ack) *)
        IF (M1_Sw AND 16#1000) = 16#1000 THEN
            M1_Cw := 16#000F;              (* drop bit 4: handshake done *)
            NewSetReq := FALSE;
        END_IF;
    ELSE
        M1_Cw := 16#000F;                  (* Operation Enabled, no new target *)
    END_IF;
ELSE
    M1_Cw := 16#0006;
END_IF;

(* the move is complete when Target Reached (bit 10) is set *)
Arrived := (M1_Sw AND 16#0400) = 16#0400;
```

A few notes on the handshake:

- Set `NewSetReq` (from your logic or the HMI) whenever you want a new positioning to start; the code above turns it into the required bit-4 pulse.
- **Set Point Acknowledge** (bit 12) tells you the drive latched the target. Only then is it safe to clear bit 4.
- **Target Reached** (bit 10) tells you the motion is finished.
- Bit 6 of the Control Word selects **absolute** (0) or **relative** (1) positioning; the examples use absolute targets.

For continuous or alternating moves (e.g. shuttling between two positions), keep bit 4 low between moves and pulse it again for each new target, waiting for bit 12 each time.

1.6. Homing

Homing (mode `6`) establishes the absolute position reference of the axis. The drive runs the procedure selected by *Homing Method* (`6098h`); the search speeds and acceleration are set by `6099h` and `609Ah`. The Mini Motor implementation supports several methods (limit/home switches, index, and the simple "set current position as zero"); see the Homing chapter of the product manual for the full list.

Like Profile Position, homing is triggered by the **New Setpoint / Homing Start** bit (bit 4) of the Control Word, but here the completion is reported by **Homing Done** — Status Word **bit 10** — while a failed homing is flagged by **Homing Error** — Status Word **bit 13**.

The sequence is:

1. select the mode (`M1_ModeOfOp := 6`) and the homing method, then wait for confirmation (`M1_ModeOfOpDisplay = 6`);
2. hold the drive in Operation Enabled (`16#000F`);
3. pulse bit 4 (`16#001F`) to start homing;
4. wait for Homing Done (bit 10), then clear bit 4.

```
(* --- Homing --- *)
M1_ModeOfOp := 6;                       (* 6060h: Homing *)
M1_HomingMethod := 37;                  (* 6098h, e.g. 37 = set current position as zero *)

IF (M1_ModeOfOpDisplay = 6) THEN
    IF HomeReq THEN
        M1_Cw := 16#001F;               (* Homing Start (bit 4) *)
    ELSE
        M1_Cw := 16#000F;               (* Operation Enabled, idle *)
    END_IF;
ELSE
    M1_Cw := 16#0006;
END_IF;

HomingDone  := (M1_Sw AND 16#0400) = 16#0400;   (* bit 10 *)
HomingError := (M1_Sw AND 16#2000) = 16#2000;   (* bit 13 *)

IF HomingDone THEN
    HomeReq := FALSE;                   (* drop bit 4: homing complete *)
END_IF;
```

Method `37` (set the current position as zero) needs no motion and completes immediately; the switch/index-based methods make the motor move to search for the reference and only then raise Homing Done. Always check **Homing Error** (bit 13) so your logic can react to a homing that could not complete.

1.7. From here on

The rest of this guide applies exactly this logic to a specific controller and development environment: installing the device description, mapping the process data, and the ready-made example project. When you read the vendor code, look for the same three ingredients — the state-machine walk to *Operation Enabled*, the mode selection, and the reference plus (for Profile Position) the New Setpoint handshake.

1.8. EtherNet/IP specifics: fixed assemblies and acyclic messaging

On EtherNet/IP the DS402 objects are exchanged through **assembly instances**. Two points make it different from CANopen/EtherCAT and are worth keeping in mind while reading the code above:

- **The cyclic process data is not freely composable.** You cannot build a custom PDO object by object; you must pick one of the **fixed assemblies** described in the *Process Data* section further down. *Default* carries the standard Profile Velocity/Torque/Position/Homing data; *A* adds digital/analog inputs and fieldbus-commanded digital outputs; *B* adds the CiA 402 touch probe; *C* the manufacturer touch probe; *D* the three-axis RMS acceleration values; *CIP Sync* carries time-stamped position set points for cyclic synchronous position, and has a chapter of its own further down.
- **There is no SDO channel.** Objects that are not part of the cyclic assembly cannot be reached with an SDO as on CANopen. To read or write them you use **acyclic (explicit) messaging**, described later in the application example.

2. EtherNet/IP

Note: available only for drivers equipped with EtherNet/IP optional board (---/---/EIP).

This document outlines the conformance of the EtherNet/IP and CIP stack implemented in the servodrive; the following documents apply unless otherwise stated:

- CIP Network library – Volume 1 (edition 3.3, november 2007)
- CIP Network library – Volume 2; EtherNet/IP adaptation of CIP (edition 1.4, november 2007)
- CiA DS402 v3.0.1.15

Furthermore the CiA DS402 adaptation to CIP are described in the relevant section.

2.1. CIP Objects

2.1.1. Overview

The servodrive implements the following objects of the CIP object model:

- CIP common objects
  - Identity object – 01h
  - Assembly object – 04h
  - TCP object – F5h
  - Ethernet object – F6h;
- Manufacturer specific objects
  - COM object – 0x64
  - PAR object – 0x65
  - MISC object – 0x66
  - CIA402 objects – 0x67

Furthermore the following ASSEMBLIES contain process data:

- O>>T: Assembly 0x64

- T>>O: Assembly 0x65
- CFG: Assembly 0x66

2.1.2. COM object - 0x64

Instance	Attribute ID	Name	Data Type	Unit meas.	Access
0x0	0x01	Revision	UINT		RO
0x1	0x01	Software edition	UINT		RO
	0x02	Software CRC	UDINT		RO
	0x10	Save parameters	UDINT		RW



Saving parameters to non-volatile memory

Writing an attribute of the PAR object only alters the RAM image of the parameter: at the next power-up the drive reloads the values stored previously.

To make a change permanent, perform a 32 bit write of the value `65766173h` (the ASCII string "SAVE") to the *Save parameters* attribute of the COM object: class `0x64`, instance `0x1`, attribute `0x10`.

2.1.3. PAR object - 0x65

Instance	Attribute ID	Name	Data Type	Unit meas.	Access
0x0	0x01	Revision	UINT		RO
0x1	0x00–0xFF	P000–P255	UINT		RW
0x2	0x00–0xFF	P256–P511	UINT		RW

Special parameters All 32 Bit parameter are divided in two 16 bits parameters of continguous numbering for the most significant and least significant bits.

These parameters are:

- P170 - Ip Addr
- P172 - Ip Mask
- P174 - Ip Gateway
- P120-135 - Multipos Ref 1 - 8

2.1.4. Misc objects - 0x66

Instance	Attribute ID	Name	Data Type	Unit meas.	Access
0x0	0x01	Revision	UINT		RO

Instance	Attribute ID	Name	Data Type	Unit meas.	Access
0x30 (48)	0x00	Analog input 0	UINT		RO
	0x01	Digital inputs	UINT		RO
	0x02	Digital outputs	UINT		RW
	0x03	Heatsink temperature	UINT	0.1 °C	RO
	0x05	Board temperature	UINT	0.1 °C	RO
	0x08	Actual Alarm	UINT	N/A	RO
	0x10	Dclink voltage	UINT	0.1 V	RO
	0x11	Current actual value	UINT	mA	RO
	0x12	Current limit for PP/PV/PT	UINT	mA	RO

2.1.5. CiA 402 Objects - 0x67

Instance	Attr. ID	Name	Data Type	Unit meas.	Access
0x0	0x01	Revision	UINT		RO
0x60 (96)	0x40	Control word	UINT		RW
	0x41	Status word	UINT		RO
	0x5A	Quick-stop option code	INT		RW
	0x5C	Disable operation option code	INT		RW
	0x5E	Fault reaction code	INT		RW
	0x60	Mode of operation	SINT		RW
	0x61	Mode of operation display	SINT		RO
	0x64	Position actual value	DINT	pulses	RO
	0x65	Following error window	DINT	pulses	RW
	0x66	Following error time	INT	ms	RW
	0x67	Position window	DINT	pulses	RW
	0x68	Position window time	INT	ms	RW
	0x6C	Velocity actual value	DINT	rpm	RO
	0x6D	Velocity window	DINT	rpm	RW
	0x6E	Velocity window time	INT	ms	RW
	0x6F	Velocity threshold	DINT	rpm	RW
	0x70	Velocity threshold time	INT	ms	RW
	0x71	Target torque	INT	thousand-th of rated	RW

Instance	Attr. ID	Name	Data Type	Unit meas.	Access
	0x72	Max torque	INT	thousand-th of rated	RW
	0x73	Max current	INT	thousand-th of rated	RW
	0x75	Motor rated current	DINT	milliA	RO
	0x76	Motor rated torque	DINT	milliNm	RO
	0x77	Torque actual value	INT	thousand-th of rated	RO
	0x78	Current actual value	INT	thousand-th of rated	RO
	0x7A	Target position	DINT	pulses	RW
	0x7B	Position Range index1: min limit index2: max limit	ARRAY[1..2] OF DINT	pulses	RW
	0x7C	Home offset	DINT	pulses	RW
	0x7D	Position limits index1: min limit index2: max limit	ARRAY[1..2] OF DINT	pulses	RW
	0x7E	Polarity	USINT		RW
	0x80	Max motor speed	DINT	rpm	RO
	0x81	Profile velocity	DINT	rpm	RW
	0x83	Profile acceleration	DINT	rpm/s	RW
	0x84	Profile deceleration	DINT	rpm/s	RW
	0x85	Quick stop deceleration	DINT	rpm/s	RW
	0x87	Torque slope	INT	thousandth of rated / s	RW
	0x98	Homing method	SINT		RW
	0x99	Homing speed index1: switch speed index2: index speed	ARRAY[1..2] OF DINT	rpm	RW
	0x9A	Homing acceleration	DINT	rpm/s	RW
	0xF2	Profile Option Code	INT		RW
	0xFF	Target velocity	DINT	rpm	RW

2.2. Process Data

_Valid from Firmware ≥ v4.036 _ There are six possible configurations of the EtherNet/IP process data.

- **Default (DFL)**: the standard process data implementing the profile velocity/torque/position/homing
- **A**: Standard process data with the possibility to read Digital and Analog inputs and command via fieldbus the Digital Out. The latter needs the parameter P015 on 10-Fieldbus to receive commands.

- **B**: A process data with the CiA 402 Touch probe functionality
- **C**: A process data with our manufacturer custom Touch Probe implementation
- **D**: A process data with acceleration three axis rms values
- **CIP Sync**: time-stamped position set points for cyclic synchronous position, synchronised with CIP Sync (PTP). It needs the NetX90 fieldbus board, firmware 5.005 or later and the CSP option

You must select the corresponding device description and set the parameter P176 to the desired value.

2.2.1. Layout Default (DFL)

The configuration for process data is the following:

Assembly 0x64 (Originator to Target, DBS/FC inputs - PLC outputs)
Assembly Length : 50 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	ControlWord	0x67	0x60	0x40	0000h	see DSP402
0x02	S32	TargetPosition	0x67	0x60	0x7A	+0	position units
0x06	U16	TorqueLimit	0x67	0x60	0x72	1000	× 0.1% Trq
0x08	S32	TargetVelocity	0x67	0x60	0xFF	+0	rpm
0x0C	S16	TargetTorque	0x67	0x60	0x71	+0	× 0.1% Trq
0x0E	U8	ModeOfOperation	0x67	0x60	0x60	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U32	ProfileVelocity	0x67	0x60	0x81	+3000	rpm
0x14	U32	ProfileAccel	0x67	0x60	0x83	+3000	rpm/s
0x18	U32	ProfileDecel	0x67	0x60	0x84	+3000	rpm/s
0x1C	S32	PositionLimit.Neg	0x67	0x60	0x7D.1	-2147483648	position units
0x20	S32	PositionLimit.Pos	0x67	0x60	0x7D.2	+2147483647	position units
0x24	S32	HomingOffset	0x67	0x60	0x7C	+0	position units
0x28	S8	HomingMethod	0x67	0x60	0x98	37	SetQuota [1]

Assembly 0x65 (Target to Originator, DBS/FC outputs - PLC inputs)
Assembly Length : 20 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	StatusWord	0x67	0x60	0x41	0250h	Motor Status [1]
0x02	S32	PositionActualValue	0x67	0x60	0x64	+0	position units
0x06	S16	TorqueActualValue	0x67	0x60	0x77	+0	× 0.1% Trq
0x08	S32	VelocityActualValue	0x67	0x60	0x6C	+0	rpm
0x0C	U16	VDclink	0x67	0x30	0x10	+0	× 0.1 V
0x0E	U8	ModeOfOperationDisplay	0x67	0x60	0x61	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U16	ActualAlarm	0x67	0x30	0x08	0	see DBS Manual

2.2.2. Layout A

The configuration for process data is the following:

Assembly 0x64 (Originator to Target, DBS/FC inputs - PLC outputs)
Assembly Length : 50 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	ControlWord	0x67	0x60	0x40	0000h	see DSP402
0x02	S32	TargetPosition	0x67	0x60	0x7A	+0	position units
0x06	U16	TorqueLimit	0x67	0x60	0x72	1000	× 0.1% Trq
0x08	S32	TargetVelocity	0x67	0x60	0xFF	+0	rpm
0x0C	S16	TargetTorque	0x67	0x60	0x71	+0	× 0.1% Trq
0x0E	U8	ModeOfOperation	0x67	0x60	0x60	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U32	ProfileVelocity	0x67	0x60	0x81	+3000	rpm
0x14	U32	ProfileAccel	0x67	0x60	0x83	+3000	rpm/s
0x18	U32	ProfileDecel	0x67	0x60	0x84	+3000	rpm/s
0x1C	S32	PositionLimit.Neg	0x67	0x60	0x7D.1	-2147483648	position units
0x20	S32	PositionLimit.Pos	0x67	0x60	0x7D.2	+2147483647	position units
0x24	S32	HomingOffset	0x67	0x60	0x7C	+0	position units

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x28	S8	HomingMethod	0x67	0x60	0x98	37	SetQuota [1]
0x29	U8	Padding Byte					
0x2A	U16	Digital output	0x67	0x30	0x02	0	bit0: DOUT1 bit1..15: not used [2]

Assembly 0x65 (Target to Originator, DBS/FC outputs - PLC inputs)
Assembly Length : 30 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	StatusWord	0x67	0x60	0x41	0250h	Motor Status [1]
0x02	S32	PositionActualValue	0x67	0x60	0x64	+0	position units
0x06	S16	TorqueActualValue	0x67	0x60	0x77	+0	× 0.1% Trq
0x08	S32	VelocityActualValue	0x67	0x60	0x6C	+0	rpm
0x0C	U16	VDclink	0x67	0x30	0x10	+0	× 0.1 V
0x0E	U8	ModeOfOperationDisplay	0x67	0x60	0x61	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U16	ActualAlarm	0x67	0x30	0x08	0	see DBS Manual
0x12	S16	Analog Input	0x67	0x30	0x00	0	-32768 … +32767
0x14	U16	Digital Input	0x67	0x30	0x01	0	Bit0: DIN1 Bit1: DIN2 Bit2: DIN3 Bit3: DIN4 Bit4: DIN5 Bit5..15: not used

2.2.3. Layout B

The configuration for process data is the following:

Assembly 0x64 (Originator to Target, DBS/FC inputs - PLC outputs)
Assembly Length : 50 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	ControlWord	0x67	0x60	0x40	0000h	see DSP402
0x02	S32	TargetPosition	0x67	0x60	0x7A	+0	position units
0x06	U16	TorqueLimit	0x67	0x60	0x72	1000	× 0.1% Trq
0x08	S32	TargetVelocity	0x67	0x60	0xFF	+0	rpm

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x0C	S16	TargetTorque	0x67	0x60	0x71	+0	× 0.1% Trq
0x0E	U8	ModeOfOperation	0x67	0x60	0x60	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U32	ProfileVelocity	0x67	0x60	0x81	+3000	rpm
0x14	U32	ProfileAccel	0x67	0x60	0x83	+3000	rpm/s
0x18	U32	ProfileDecel	0x67	0x60	0x84	+3000	rpm/s
0x1C	S32	PositionLimit.Neg	0x67	0x60	0x7D.1	-2147483648	position units
0x20	S32	PositionLimit.Pos	0x67	0x60	0x7D.2	+2147483647	position units
0x24	S32	HomingOffset	0x67	0x60	0x7C	+0	position units
0x28	S8	HomingMethod	0x67	0x60	0x98	37	SetQuota [1]
0x29	U8	Padding Byte					
0x2A	U16	Digital output	0x67	0x30	0x02	0	bit0: DOUT1 bit1..15: not used [2]
0x2C	U16	Touch Probe Function	0x67	0x60	0xB8	0	[1]

Assembly 0x65 (Target to Originator, DBS/FC outputs - PLC inputs)
Assembly Length : 40 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	StatusWord	0x67	0x60	0x41	0250h	Motor Status [1]
0x02	S32	PositionActualValue	0x67	0x60	0x64	+0	position units
0x06	S16	TorqueActualValue	0x67	0x60	0x77	+0	× 0.1% Trq
0x08	S32	VelocityActualValue	0x67	0x60	0x6C	+0	rpm
0x0C	U16	VDclink	0x67	0x30	0x10	+0	× 0.1 V
0x0E	U8	ModeOfOperationDisplay	0x67	0x60	0x61	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U16	ActualAlarm	0x67	0x30	0x08	0	see DBS Manual
0x12	S16	Analog Input	0x67	0x30	0x00	0	-32768 … +32767

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x14	U16	Digital Input	0x67	0x30	0x01	0	Bit0: DIN1 Bit1: DIN2 Bit2: DIN3 Bit3: DIN4 Bit4: DIN5 Bit5..15: not used
0x16	U16	Touch Probe Status	0x67	0x60	0xB9	0	[1]
0x18	S32	Touch Probe Rising Edge position	0x67	0x60	0xBA	0	position units
0x1C	S32	Touch Probe Falling Edge position	0x67	0x60	0xBB	0	position units

2.2.4. Layout C

The configuration for process data is the following:

Assembly 0x64 (Originator to Target, DBS/FC inputs - PLC outputs)
Assembly Length : 60 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	ControlWord	0x67	0x60	0x40	0000h	see DSP402
0x02	S32	TargetPosition	0x67	0x60	0x7A	+0	position units
0x06	U16	TorqueLimit	0x67	0x60	0x72	1000	× 0.1% Trq
0x08	S32	TargetVelocity	0x67	0x60	0xFF	+0	rpm
0x0C	S16	TargetTorque	0x67	0x60	0x71	+0	× 0.1% Trq
0x0E	U8	ModeOfOperation	0x67	0x60	0x60	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U32	ProfileVelocity	0x67	0x60	0x81	+3000	rpm
0x14	U32	ProfileAccel	0x67	0x60	0x83	+3000	rpm/s
0x18	U32	ProfileDecel	0x67	0x60	0x84	+3000	rpm/s
0x1C	S32	PositionLimit.Neg	0x67	0x60	0x7D.1	-2147483648	position units
0x20	S32	PositionLimit.Pos	0x67	0x60	0x7D.2	+2147483647	position units
0x24	S32	HomingOffset	0x67	0x60	0x7C	+0	position units

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x28	S8	HomingMethod	0x67	0x60	0x98	37	SetQuota [1]
0x29	U8	Padding Byte					
0x2A	U16	Digital output	0x67	0x30	0x02	0	bit0: DOUT1 bit1..15: not used [2]
0x2C	S32	Touch Probe positioning offset	0x67	0x30	0x20	0	Position Units
0x2C	S32	Touch Probe positioning modulo	0x67	0x30	0x21	0	Position Units

Assembly 0x65 (Target to Originator, DBS/FC outputs - PLC inputs)
Assembly Length : 30 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	StatusWord	0x67	0x60	0x41	0250h	Motor Status [1]
0x02	S32	PositionActualValue	0x67	0x60	0x64	+0	position units
0x06	S16	TorqueActualValue	0x67	0x60	0x77	+0	× 0.1% Trq
0x08	S32	VelocityActualValue	0x67	0x60	0x6C	+0	rpm
0x0C	U16	VDclink	0x67	0x30	0x10	+0	× 0.1 V
0x0E	U8	ModeOfOperationDisplay	0x67	0x60	0x61	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U16	ActualAlarm	0x67	0x30	0x08	0	see DBS Manual
0x12	S16	Analog Input	0x67	0x30	0x00	0	-32768 … +32767
0x14	U16	Digital Input	0x67	0x30	0x01	0	Bit0: DIN1 Bit1: DIN2 Bit2: DIN3 Bit3: DIN4 Bit4: DIN5 Bit5..15: not used

2.2.5. Layout D

The configuration for process data is the following:

Assembly 0x64 (Originator to Target, DBS/FC inputs - PLC outputs)
Assembly Length : 50 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	ControlWord	0x67	0x60	0x40	0000h	see DSP402
0x02	S32	TargetPosition	0x67	0x60	0x7A	+0	position units
0x06	U16	TorqueLimit	0x67	0x60	0x72	1000	× 0.1% Trq
0x08	S32	TargetVelocity	0x67	0x60	0xFF	+0	rpm
0x0C	S16	TargetTorque	0x67	0x60	0x71	+0	× 0.1% Trq
0x0E	U8	ModeOfOperation	0x67	0x60	0x60	1	Profile [1]
0x0F	U8	Padding Byte					
0x10	U32	ProfileVelocity	0x67	0x60	0x81	+3000	rpm
0x14	U32	ProfileAccel	0x67	0x60	0x83	+3000	rpm/s
0x18	U32	ProfileDecel	0x67	0x60	0x84	+3000	rpm/s
0x1C	S32	PositionLimit.Neg	0x67	0x60	0x7D.1	-2147483648	position units
0x20	S32	PositionLimit.Pos	0x67	0x60	0x7D.2	+2147483647	position units
0x24	S32	HomingOffset	0x67	0x60	0x7C	+0	position units
0x28	S8	HomingMethod	0x67	0x60	0x98	37	SetQuota [1]
0x29	U8	Padding Byte					
0x2A	U16	Digital output	0x67	0x30	0x02	0	bit0: DOUT1 bit1..15: not used [2]

Assembly 0x65 (Target to Originator, DBS/FC outputs - PLC inputs)
Assembly Length : 30 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	U16	StatusWord	0x67	0x60	0x41	0250h	Motor Status [1]
0x02	S32	PositionActualValue	0x67	0x60	0x64	+0	position units
0x06	S16	TorqueActualValue	0x67	0x60	0x77	+0	× 0.1% Trq
0x08	S32	VelocityActualValue	0x67	0x60	0x6C	+0	rpm
0x0C	U16	VDclink	0x67	0x30	0x10	+0	× 0.1 V
0x0E	U8	ModeOfOperationDisplay	0x67	0x60	0x61	1	Profile [1]

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x0F	U8	Padding Byte					
0x10	U16	ActualAlarm	0x67	0x30	0x08	0	see DBS Manual
0x12	S16	Analog Input	0x67	0x30	0x00	0	-32768 … +32767
0x14	U16	Digital Input	0x67	0x30	0x01	0	Bit0: DIN1 Bit1: DIN2 Bit2: DIN3 Bit3: DIN4 Bit4: DIN5 Bit5..15: not used
0x16	U16	Acceleration rms	0x67	0x30	0x31		
0x19	U16	Acceleration rms	0x67	0x30	0x32		
0x1A	U16	Acceleration rms	0x67	0x30	0x33		

2.2.6. Layout CIP Sync

Valid from Firmware ≥ v5.005, NetX90 fieldbus board and CSP option

P176 = 5, device description `DBS55_CIP_sync.eds`. The PLC sends with each target position the time it refers to, *SetpointTime*, and the drive interpolates between the last three pairs received; the drive time-stamps its actual values with *PositionTime*. Both times are in nanoseconds of the clock shared through CIP Sync (PTP), which has to be enabled on the controller. The Add-On Instructions for this layout and the configuration of Studio 5000 are described in the EtherNet/IP set up guide.

The configuration for process data is the following:

Assembly 0x64 (Originator to Target, DBS/FC inputs - PLC outputs)
Assembly Length : 30 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	S64	SetpointTime	0x67	0x30	0x70	0	ns, PTP time
0x08	U16	ControlWord	0x67	0x60	0x40	0000h	see DSP402
0x0A	U8	ModeOfOperation	0x67	0x60	0x60	8	8 = Cyclic Synchronous Position, 6 = Homing
0x0B	U8	Padding Byte					
0x0C	S32	TargetPosition	0x67	0x60	0x7A	+0	position units
0x10	S32	VelocityOffset	0x67	0x60	0xB1	+0	rpm
0x14	S32	HomingOffset	0x67	0x60	0x7C	+0	position units
0x18	S8	HomingMethod	0x67	0x60	0x98	37	SetQuota [1]
0x19	U8	Padding Byte					

Assembly 0x65 (Target to Originator, DBS/FC outputs - PLC inputs)
Assembly Length : 40 Bytes

Offset	Type	Attribute	Class	Instance	Attr. ID	Default	Meaning
0x00	S64	PositionTime	0x67	0x30	0x71	0	ns, PTP time
0x08	U16	StatusWord	0x67	0x60	0x41	0250h	Motor Status [1]
0x0A	U8	ModeOfOperationDisplay	0x67	0x60	0x61	8	Profile [1]
0x0B	U8	Padding Byte					
0x0C	S32	PositionActualValue	0x67	0x60	0x64	+0	position units
0x10	S32	VelocityActualValue	0x67	0x60	0x6C	+0	rpm
0x14	S32	FollowingErrorActualValue	0x67	0x60	0xF4	+0	position units
0x18	S16	TorqueActualValue	0x67	0x60	0x77	+0	× 0.1% Trq
0x1A	U16	ActualAlarm	0x67	0x30	0x08	0	see DBS Manual
0x1C	U16	VDclink	0x67	0x30	0x10	+0	× 0.1 V
0x1E	U16	Digital Input	0x67	0x30	0x01	0	Bit0..4: DIN1..DIN5
0x20	U16	SyncStatus	0x67	0x30	0x72	0	bit0: PTP synchronised bit1: time valid bit2: sync0 ok bit3: locked to the time base bit4: set point time received

[1] see DSP402

[2] P015=10 required for fieldbus operation

3. Setting Up the Project

3.1. Tools

Program Example (no AOI)

Program Example (AOI, DS402)

AOI Download (DS402)

AOI + Program Download (Rockwell-like) V1.1+

CIP SYNC AOI and Example

DBS/FC BSI Interface Software

3.2. Overview

This document shows the creation of a Mini Motor Integrated Drive application using the EtherNet/IP communication interface; a Rockwell CompactLogix controller is configured and an application is written into the Studio 5000 development environment.

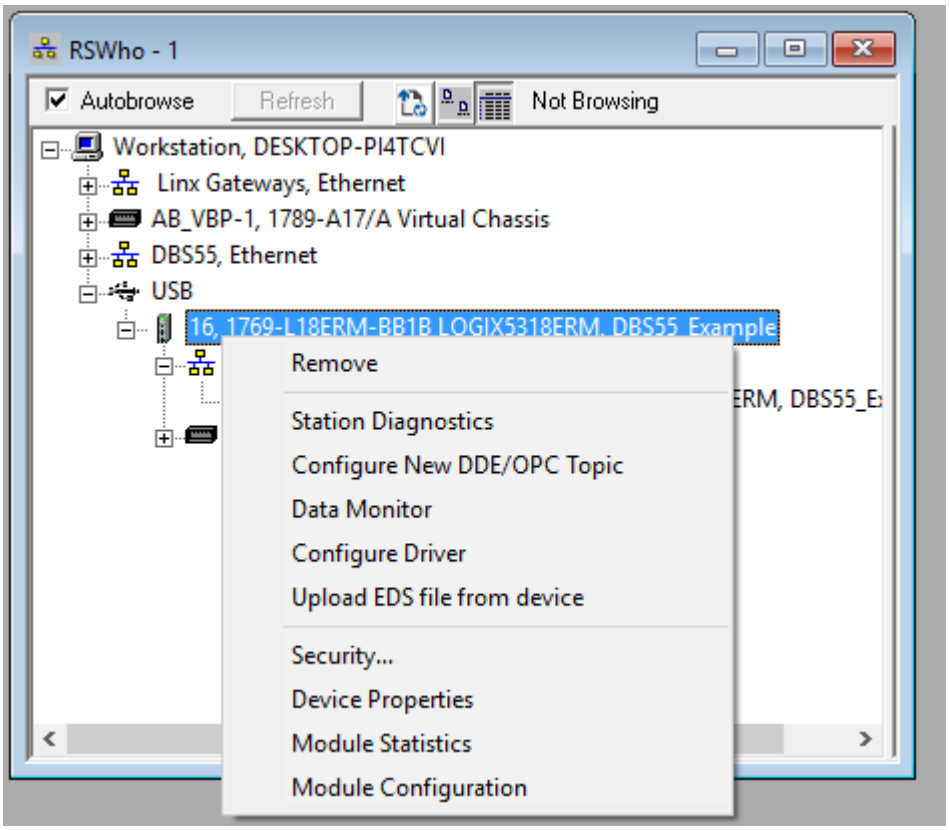
References:

- CompactLogix L18ERM
- DBS/FC firmware version 5.005
- Studio 5000, version 31.01.00

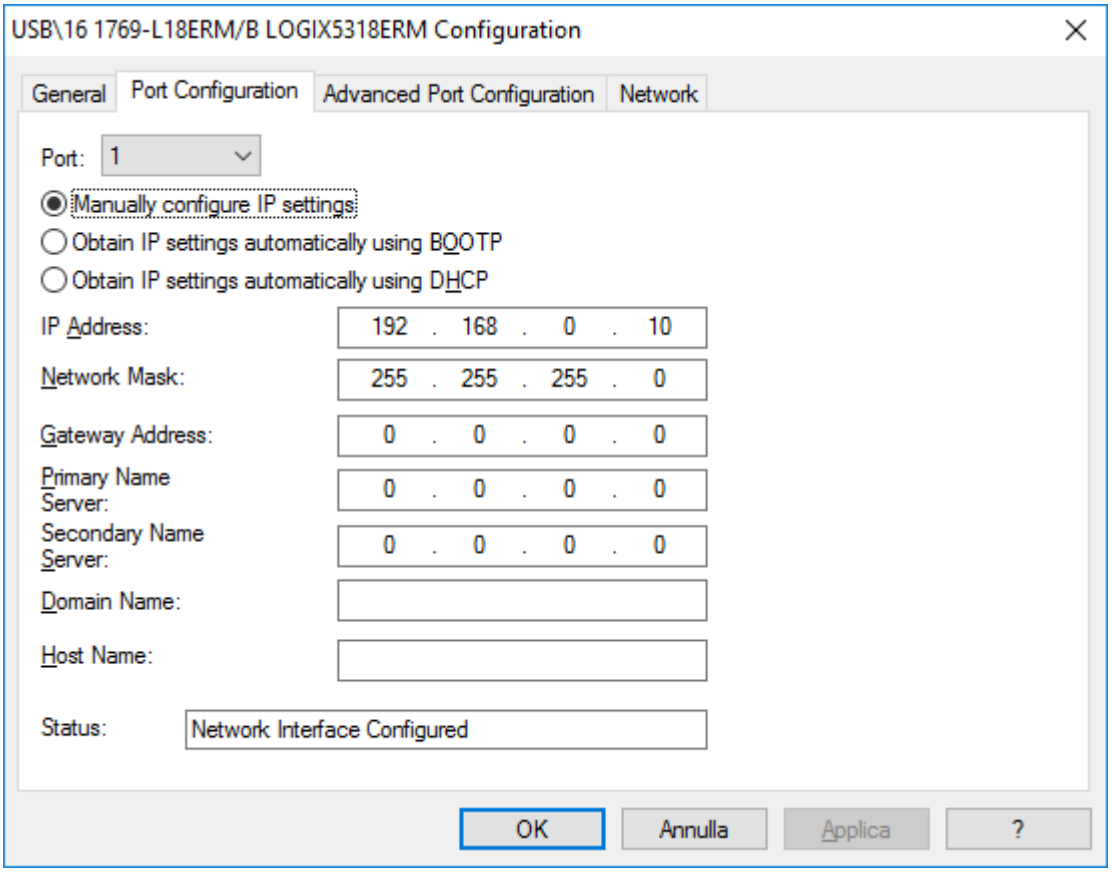


3.3. Configure Controller Address

- To set the controller address, start RSLinx
- Select Communication → RSWho and open the controller node (in the example the controller is connected via USB); then right click and select *Module Configuration*



- In the configuration dialog, select *Port Configuration* (1), then *Manually configure IP settings* (2) and insert address and mask (3); in the example 192.168.0.10 is set
- Confirm with OK.



3.4. Configure the Motor Address

The drive leaves the factory with a **static** address, which is the one you will find on a brand new unit:

Parameter	Name and meaning	Factory value
P170	*IpAddr* — IP address of the drive	192.168.250.100
P172	*IpMask* — subnet mask	255.255.0.0
P174	*IpGateway* — default gateway	0.0.0.0
P215	*IpMode* — how the address is assigned	0 — Static

Three addressing modes are available, selected with P215:

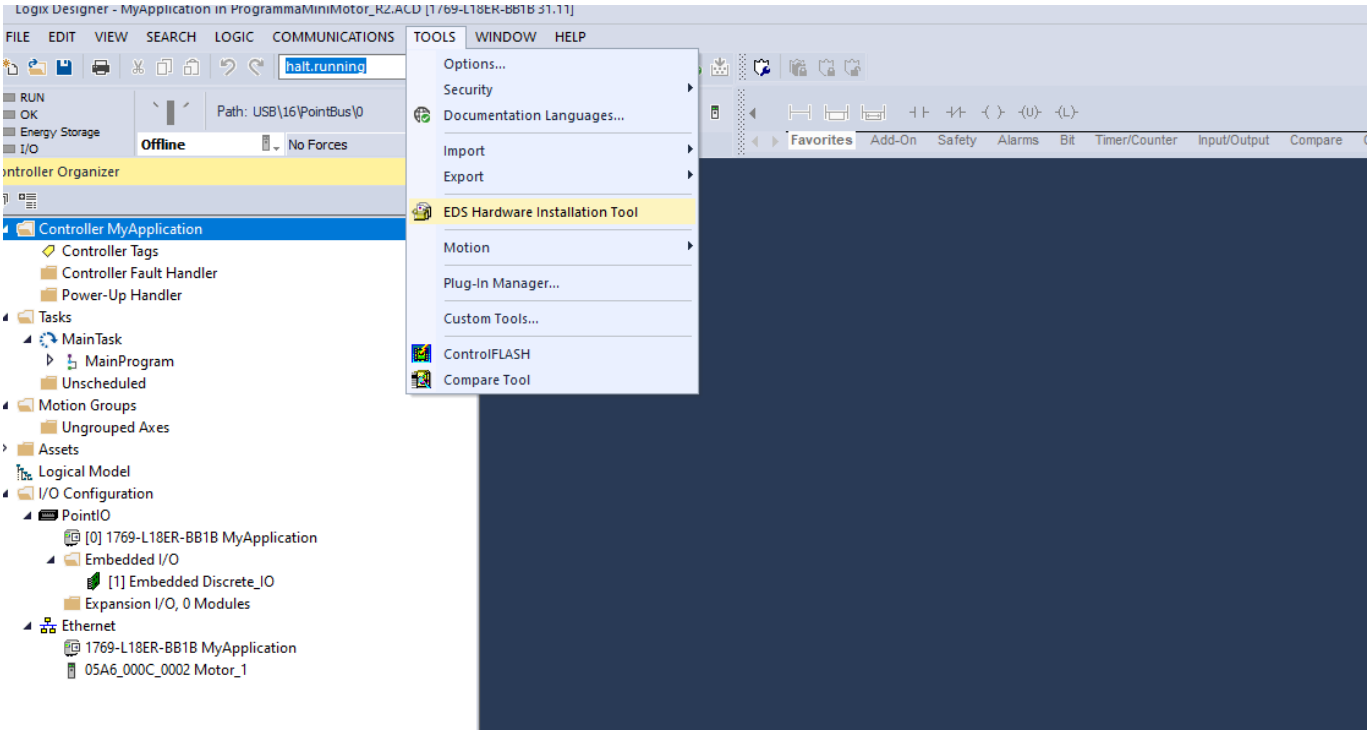
Mode	How it works
Static, P215 = 0	Write P170, P172 and P174 with the BSI software through the service port, then power cycle the drive. BSI saves them for you
DHCP, P215 = 1	The drive asks a DHCP server at every power up and ignores P170 to P174. It shows up in the Rockwell *BOOTP/DHCP Commissioning Tool* with its MAC address
BOOTP, P215 = 2	The drive asks for its address via BOOTP, for example from the Rockwell *BOOTP/DHCP Commissioning Tool*, where it shows up with its MAC address. Assign the address, then press *Disable BOOTP/DHCP*: the drive saves the address in P170 to P174 and switches by itself to static (P215 = 0)



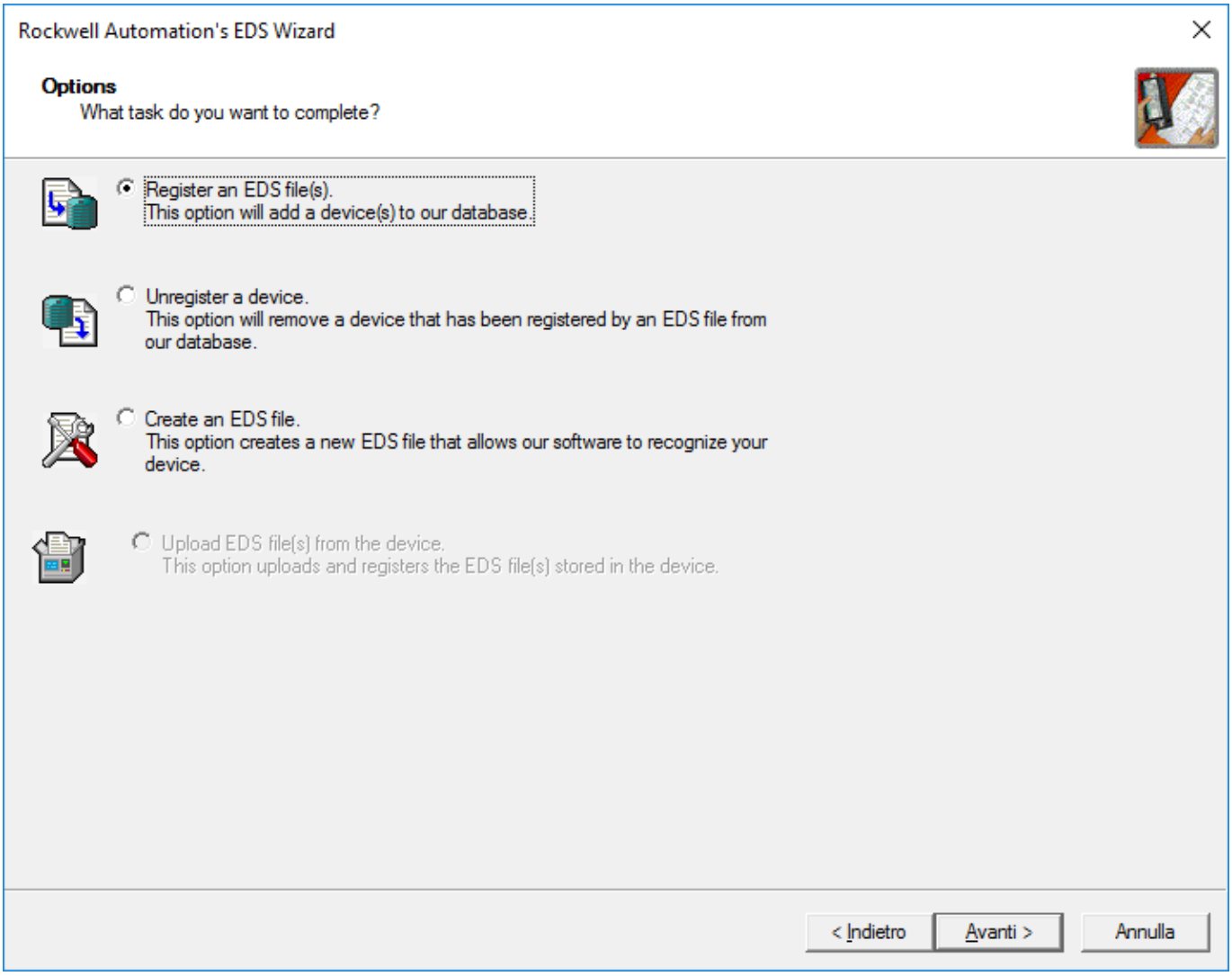
The Studio 5000 module points to one fixed address. In DHCP, reserve the lease for the MAC address of the drive, or the connection is lost at the next reboot. Address changes made by tools in DHCP are not saved. If you do not need DHCP, static is simpler, and BOOTP is the quickest way to set it without BSI. Either way the drive must be in the same subnet as the controller. P215 exists from firmware 4.056, the BOOTP mode from firmware 5.005.

3.5. Install EDS

- In Studio 5000 click on Tools → EDS Hardware Installation Tool

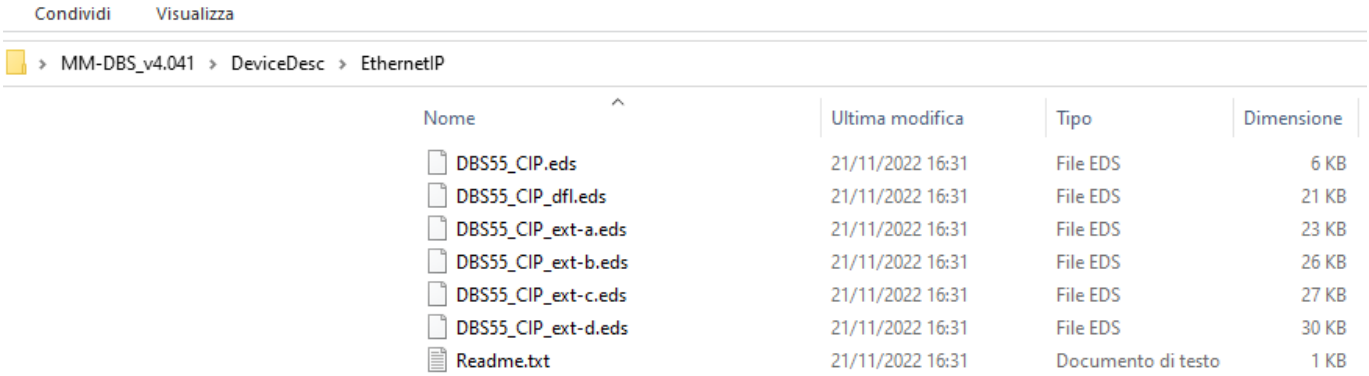


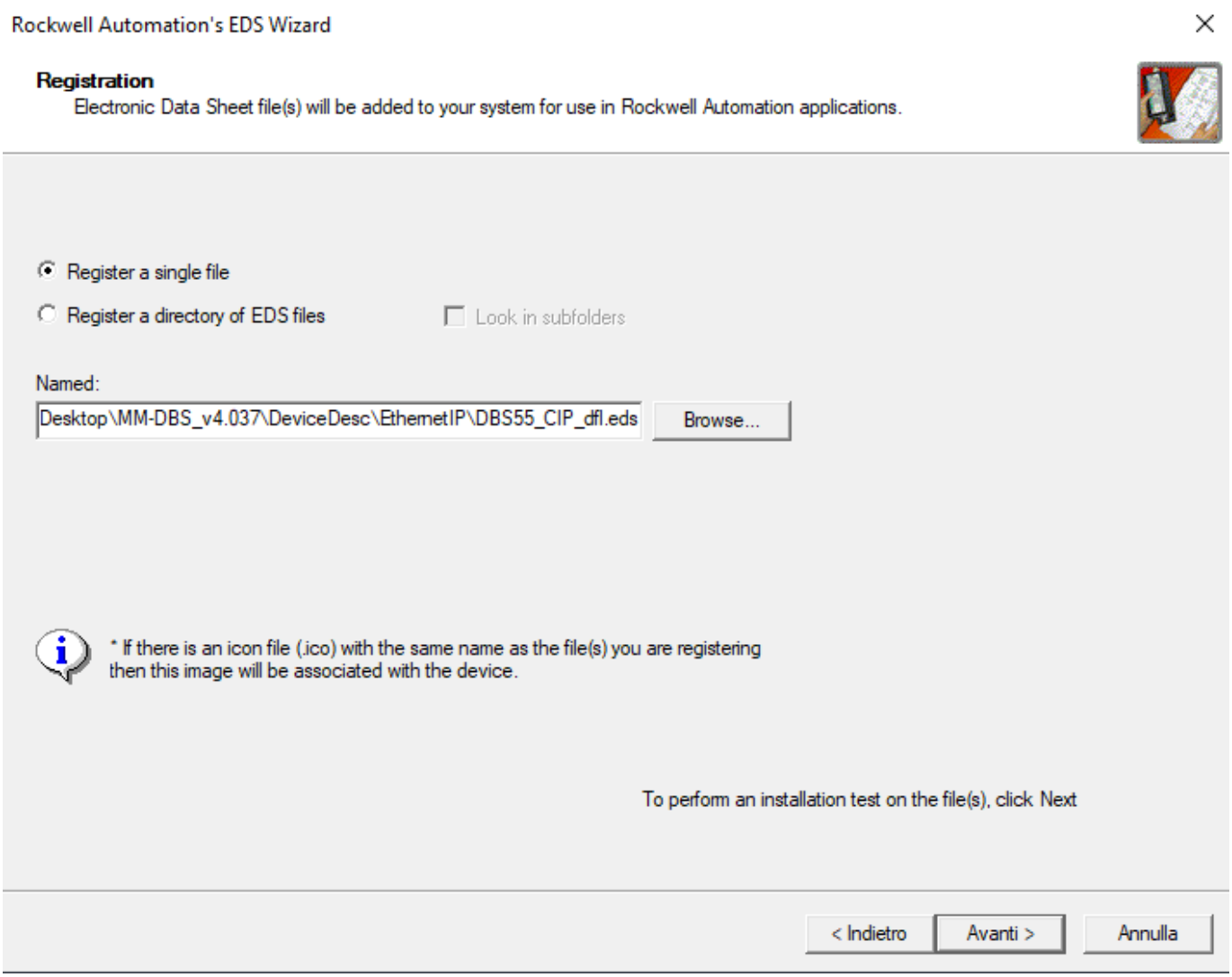
- Register an EDS file



- Browse
- Open the BSI software revision\DeviceDesc\EthernetIP folder and select the EIP layout you desire. For this example we will select DBS55_CIP_dfl.eds. Install only the one you want to use.

> The EDS file must match the process data layout selected on the drive with P176. The DS402 and Rockwell-like blocks use the **Default (DFL)** layout, so `DBS55_CIP_dfl.eds` with P176 = 0; the CIP Sync blocks use `DBS55_CIP_sync.eds` with P176 = 5.

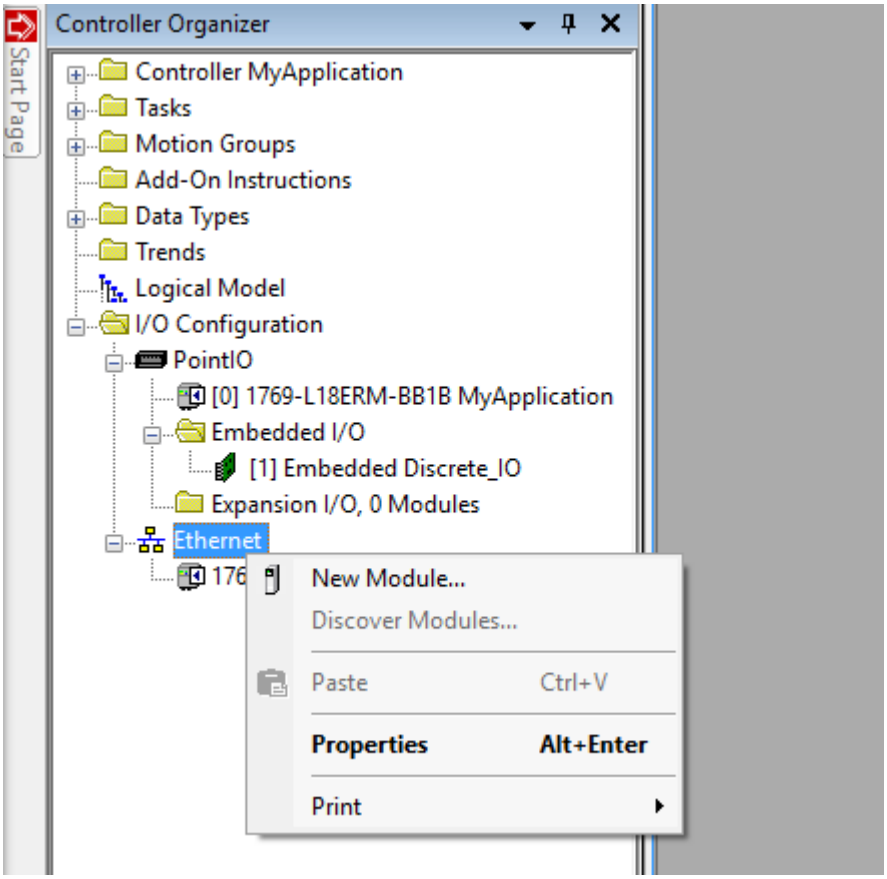




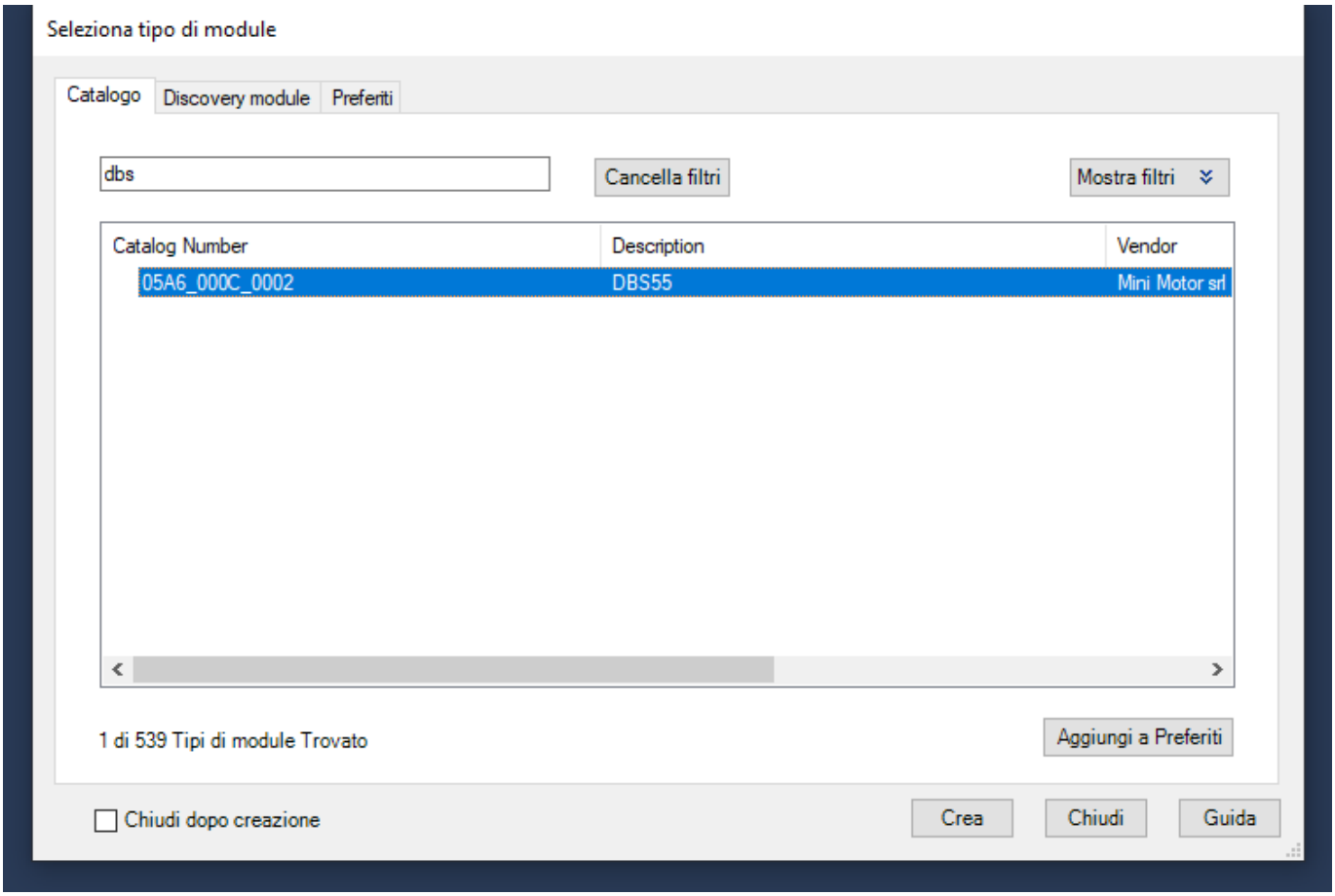
- Next →, Next →, Next →; click on Finish to complete the insertion.

3.6. Add DBS Slave to a Project

- Right click on *I/O Configuration* > *Ethernet* and select *New Module*

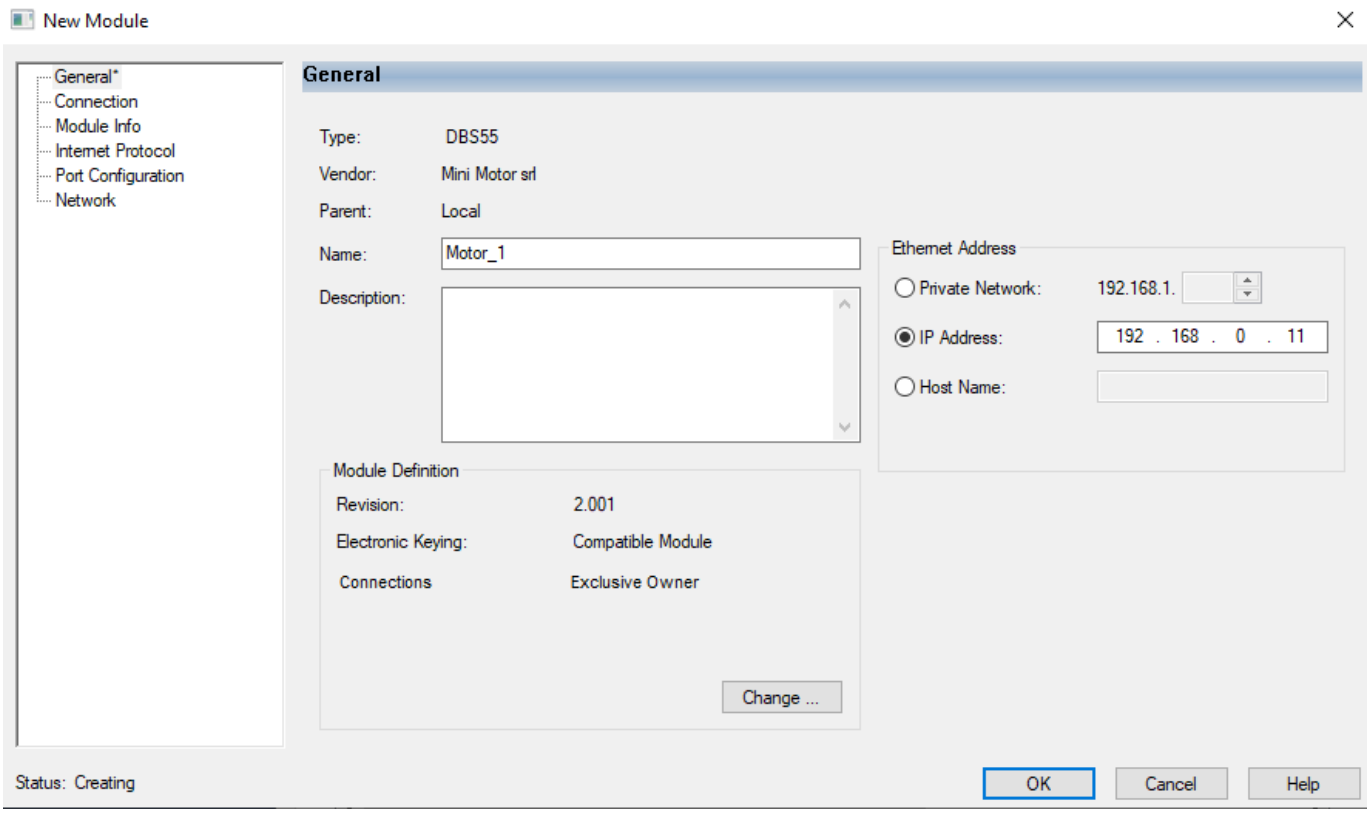


- Type dbs in the filter and select *05A6_000C_0002 – DBS55*, then click on Create

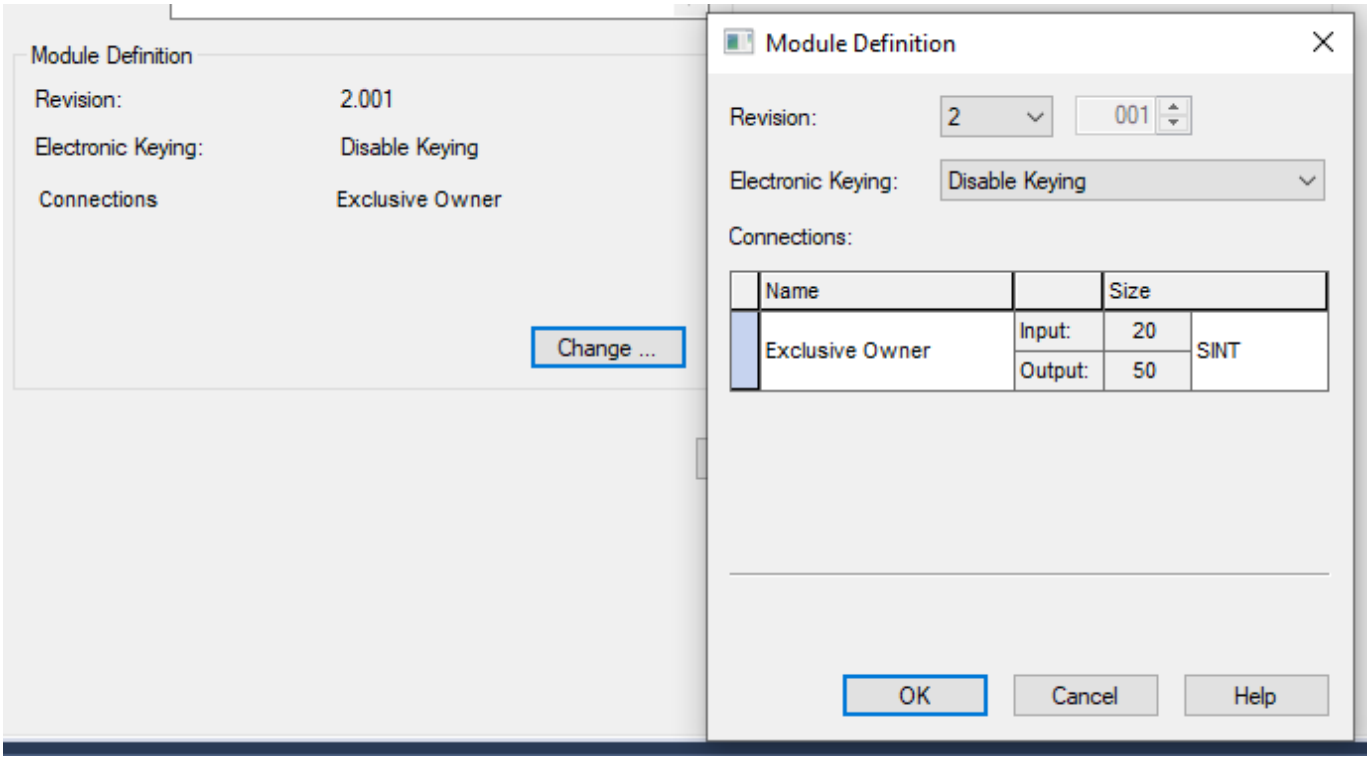


- In the *New Module* window enter the device name and the IP address of the device, then finally click on Change.

> The address you type here is the one the drive actually answers to, so it must match what you configured in Configure the Motor Address and be in the same subnet as the controller.



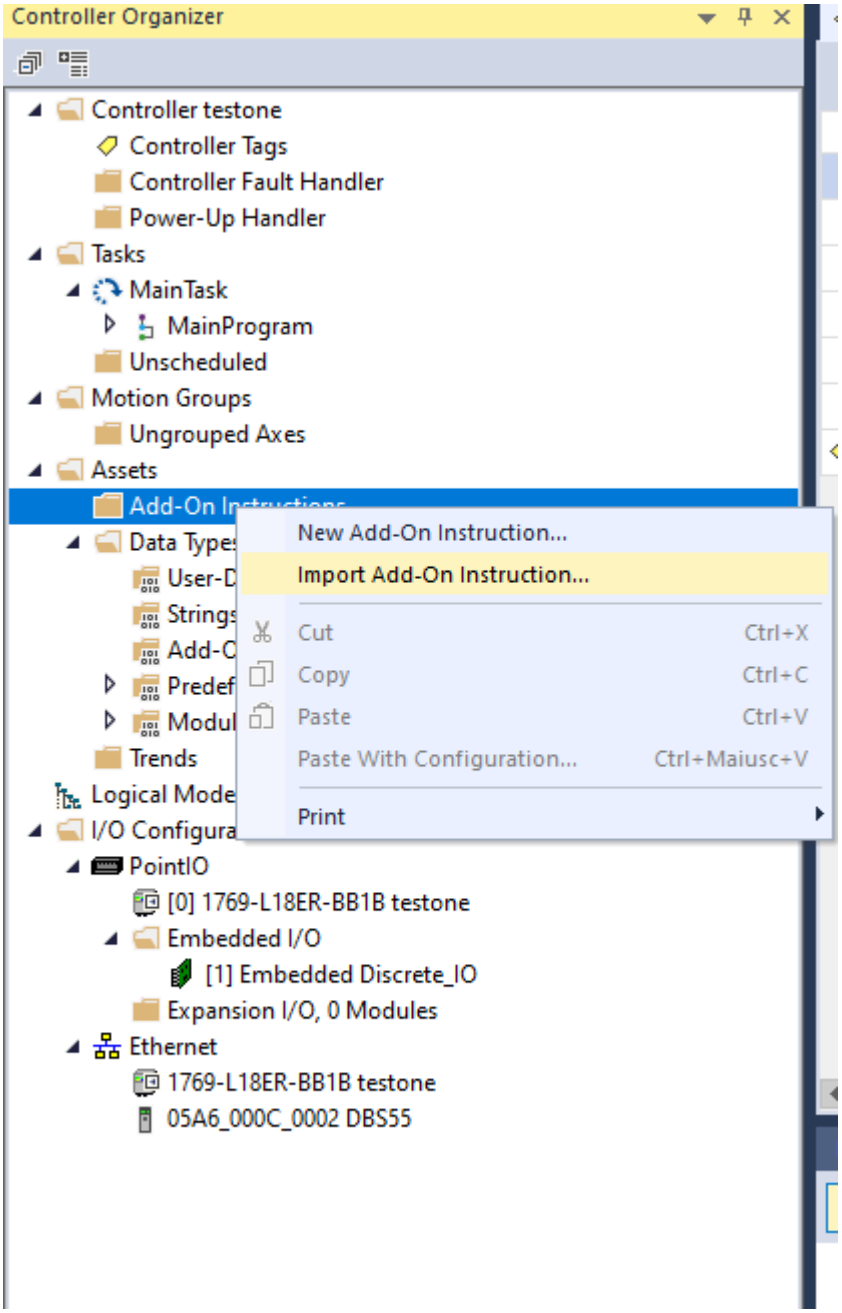
- In the module definition window, in the *Electronic Keying* field select *Disable Keying*
- Confirm the module definition dialog and finally click on Create, then close the dialog



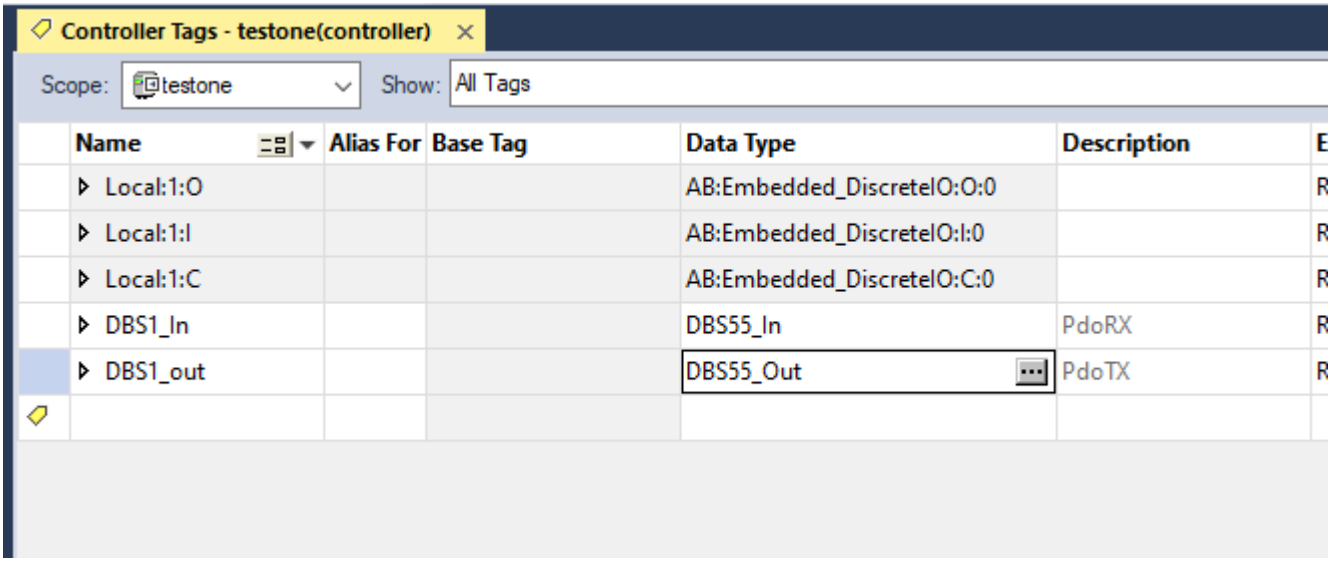
3.7. AOI Import

- Import the DBS AOI you need for your project. Go to *Assets → Add-On Instructions → Right Click → select which AOI to import*.

Each `.L5X` file contains one single Add-On Instruction, so importing one block never touches the others. The DS402 and Rockwell-like blocks carry the `DBS55_In` and `DBS55_Out` definitions with them, the CIP Sync blocks `DBS55S_In`, `DBS55S_Out` and `MM_CS_Time`: the first import creates the data types, the following ones find them already there.



- After having imported an AOI you should find its User Defined Data Types: two for the DS402 and Rockwell-like families, three for the CIP Sync one.
- Create for each motor a variable of both types.



3.8. Choosing the AOI Family

Three families of Add-On Instructions drive the same motor over the same connection. What changes is who computes the trajectory, the drive or the PLC.

Family	Trajectory computed by	Layout, P176	Use it when
DS402, `MM_*_dfl`	The drive, in the CiA 402 profile modes	Default, 0	You want the drive to do the work: the PLC asks for a position, a speed or a torque and the drive builds the ramps. This is the recommended family
Rockwell-like, `MM_MSO`, `MM_MAM`…	The drive, in the same profile modes	Default, 0	Your team writes motion code for Rockwell drives: same functions as the DS402 family, with the names, the pins and the Done/Busy/Error handshake of the Logix motion instructions
CIP Sync, `MM_CS*`	The motion planner of the controller	CIP Sync, 5	The axis has to be part of a Logix motion group: a virtual axis is moved with MAJ, MAM, MAG or a cam, and the drive follows it in cyclic synchronous position. Requires the CSP option on the drive

The DS402 and Rockwell-like blocks share the data types `DBS55_In` (to the drive) and `DBS55_Out` (from the drive) and can be mixed on one axis. The CIP Sync blocks use another layout with their own data types: P176 decides which one a drive is on.

Unless stated otherwise, the DS402 and Rockwell-like blocks use the units of the DS402 process data:

- speeds in rpm, accelerations and decelerations in rpm/s;
- torques in thousandths of the motor rated torque, so 1000 = 100 % of rated torque;
- positions in encoder pulses, or in the user unit if you have scaled it with P160, P161 and P162 (see the main DBS/FC manual).

The CIP Sync blocks work in encoder pulses on the drive side and in the units of the virtual axis on the motion group side; the conversion between the two is a parameter of MM_CSVA.

3.9. Which Revision This Guide Describes

The revision of each block is shown in Studio 5000 under *Assets* → *Add-On Instructions*. This guide describes the revisions below: older blocks may lack some of the pins described.

Block	Revision	Notes
MM_MSO, MM_MSF, MM_MAFR, MM_MAM, MM_MAJ, MM_MAH, MM_MAS	1.2	Rockwell-like command blocks. 1.1 added *Timeout* and *ErrorCode* and corrected the defects listed in Updating the Rockwell-Like AOI from Revision 1.0; 1.2 has pin descriptions in English, no functional change
MM_MSO_ST … MM_MAS_ST	1.2	Optional variant of the same seven blocks, written in Structured Text. Same pins, same behaviour
MM_MAUI, MM_MAUO	1.3	Rockwell-like process data blocks. No functional change from 1.1
MM_MSL	1.0	New Rockwell-like block that writes the software position limits, see MM_MSL
MM_ReadData_dfl, MM_SendData_dfl, MM_Profile_*_dfl, MM_Status_dfl	1.1	DS402 family, unchanged
MM_Reset_Error_dfl	1.0	DS402 family, unchanged
MM_CST	1.3	CIP Sync family, first release
MM_CSVA	1.2	CIP Sync family, first release
MM_CSI, MM_CSO, MM_CSSTAT, MM_CSPWR, MM_CSHOME	1.1	CIP Sync family, first release



Updating the Rockwell-like blocks from revision 1.0 changes the polarity of *MoveType* on MM_MAM. Read Updating the Rockwell-Like AOI from Revision 1.0 before updating a running machine.

4. DS402 AOI

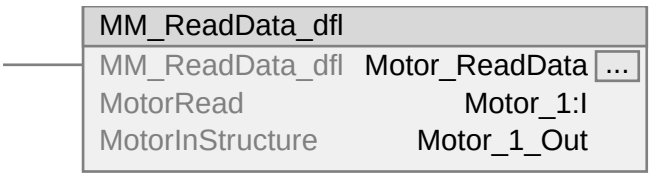
The recommended family. Each Profile block holds the drive in one CiA 402 mode of operation and the drive generates the trajectory: the PLC only sets targets and limits.

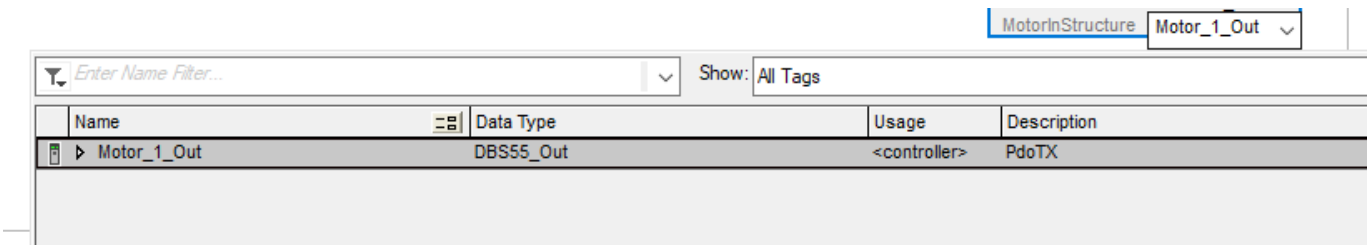
Download: AOI (DS402) and Program Example (AOI, DS402).

> This family and the Rockwell-like one work on the **Default (DFL)** layout: `DBS55_CIP_dfl.eds` and P176 = 0. With any other layout the data does not line up and the blocks do not work.

4.1. MM_ReadData_dfl

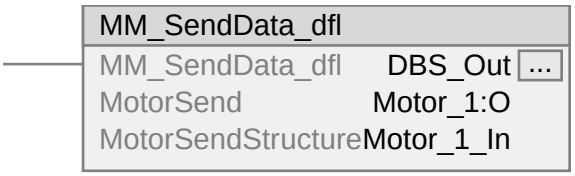
The first block of the scan for each motor. It copies the input assembly `Motor_x:I` into a `DBS55_Out` structure, which all the other blocks read.

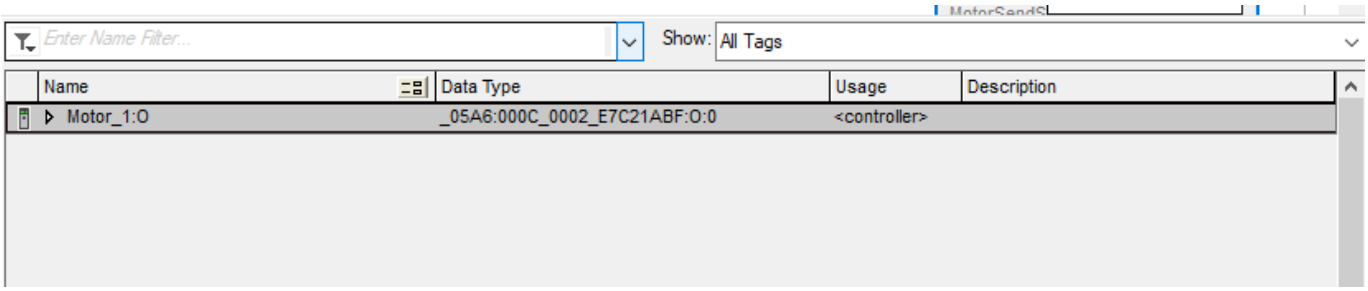




4.2. MM_SendData_dfl

The last block of the scan. It copies the `DBS55_In` structure, written by all the other blocks, into the output assembly `Motor_x:O`.

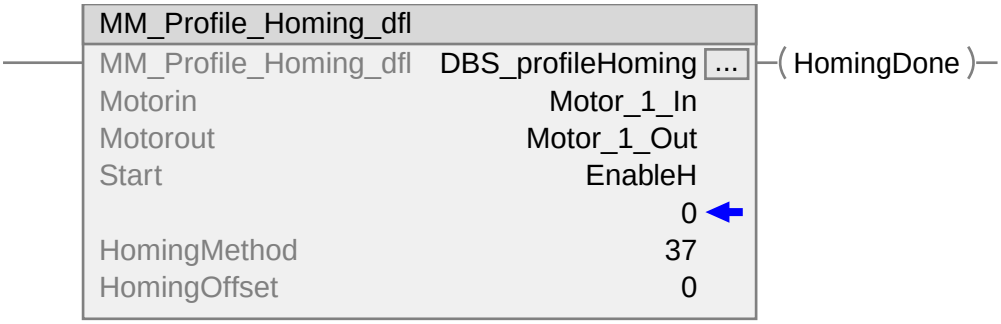




4.3. MM_Profile_Homing_dfl

Runs the CiA 402 homing of the drive (mode 6).

Pin	Meaning
Motorin, Motorout	`DBS55_In` and `DBS55_Out` of the motor
Start	Starts the homing
HomingMethod	CiA 402 homing method, see the DBS/FC manual. 37 is homing in place
HomingOffset	Position assigned at the end of the homing
HomingDone	Homing completed



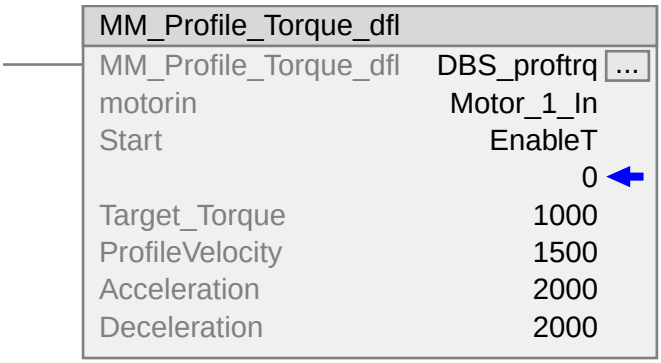
4.4. MM_Profile_Torque_dfl

Holds the drive in Profile Torque (mode 4) as long as *Start* is true.

Pin	Meaning
Target_Torque	Torque set point, thousandths of rated
ProfileVelocity	Maximum speed allowed while delivering the torque, rpm
Acceleration, Deceleration	Ramps, rpm/s



The help text inside the block also lists *PoslimPos* and *PoslimNeg*: they do not exist. The table above is the interface to trust.



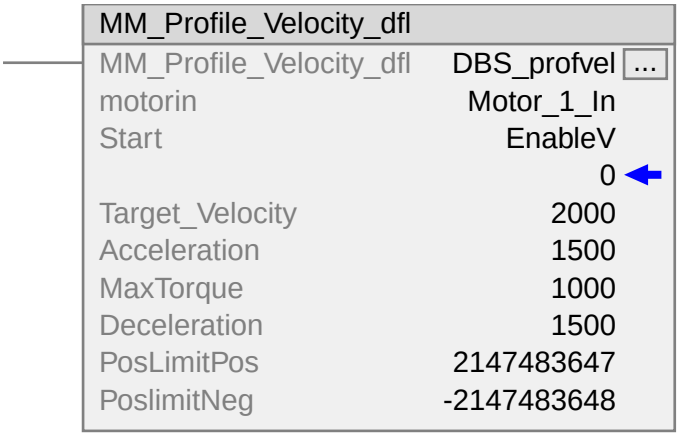
4.5. MM_Profile_Velocity_dfl

Holds the drive in Profile Velocity (mode 3) as long as *Start* is true.

Pin	Meaning
Target_Velocity	Speed set point, rpm
Acceleration, Deceleration	Ramps, rpm/s
MaxTorque	Torque limit, thousandths of rated
PosLimitPos, PoslimitNeg	Software position limits. 2147483647 and -2147483648 disable them



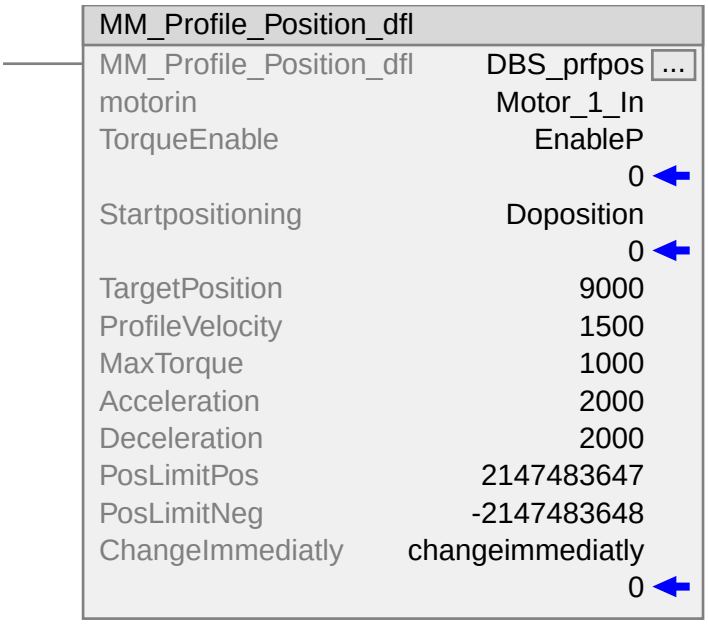
The two limits default to 0, and with 0 and 0 the motor does not move. Write the disabled values, or the limits you want, before the first start. The same applies to MM_Profile_Position_dfl.



4.6. MM_Profile_Position_dfl

Positioning in Profile Position (mode 1).

Pin	Meaning
TorqueEnable	Enables the drive, which holds its position
Startpositioning	Rising edge: starts the move
TargetPosition	Target, pulses or scaled units
ProfileVelocity, Acceleration, Deceleration	Trajectory, rpm and rpm/s
MaxTorque	Torque limit, thousandths of rated
PosLimitPos, PosLimitNeg	Software position limits, as in MM_Profile_Velocity_dfl
ChangeImmediatly	1: a new target replaces the move in progress. 0: it is queued after it



4.7. MM_Status_dfl

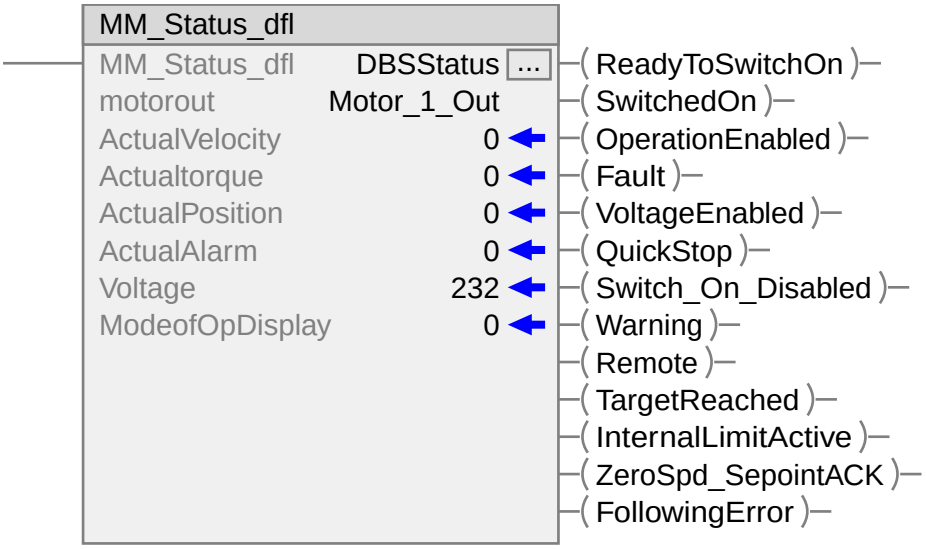
Quick view of the drive from its `DBS55_Out`: speed, torque, position, alarm, DC voltage, mode of operation and the status word bits. It works in a Rockwell-like project as well.



ZeroSpd_SepointACK is status word bit 12, typo included: *Setpoint acknowledge* in Profile Position, *Speed is zero* in Profile Velocity, *Homing attained* in Homing.

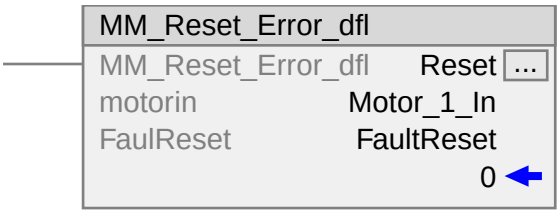
The Profile blocks are modes and have no *Done*: the end of a movement is read here.

Mode	Movement finished when
Profile Position	*TargetReached* high and *ZeroSpd_SepointACK* back low after the handshake
Profile Velocity	*TargetReached* high: the speed has been reached
Homing	*ZeroSpd_SepointACK* high (*Homing attained*)



4.8. MM_Reset_Error_dfl

Resets a drive fault on the rising edge of *FaulReset*. The fault state is shown by MM_Status_dfl.



5. Rockwell-Like AOI

The same functions as the DS402 family, with the names, the pins and the handshake of the Logix motion instructions: MSO, MAM, MAJ and so on. If your team already writes motion code for Rockwell drives, you never touch the control word and you do not need the CiA 402 state machine.

Download: AOI and Program Example (Rockwell-like).

The blocks use the same Default layout and the same `DBS55_In` and `DBS55_Out` data types as the DS402 family, so the two families can be mixed on one axis.

5.1. Common Behaviour

Topic	Behaviour
Axis_In, Axis_Out	`DBS55_In` is what goes to the drive, `DBS55_Out` what comes back. Every command block takes both
Trigger	Rising edge of the rung. Holding the rung does not repeat the command: drop it and raise it again, which is also what re-arms the block
Busy, Done, Error	*Busy* while the sequence runs, then *Done* or *Error*, held until the rung drops
Timeout, ErrorCode	*Timeout* (ms, default 5000) covers the CiA 402 transitions and the set-point handshake, not the travel: a move that never ends leaves the block *Busy*, not in *Error*. MM_MAH is the exception, see its section
Inputs	Read at every scan until the sequence ends, so MM_MAJ can change speed on the fly
Scan order	MM_MAUI first, command blocks and MM_MSL in the middle, MM_MAUO last
One command per axis	There is no arbitration: two active blocks on the same axis overwrite each other. Chain commands on *Done*
Connection loss	The blocks only see the data: if the drive goes offline `DBS55_Out` keeps the last values. Read the connection with a GSV on the MODULE object

ErrorCode	Meaning
0	No error
1	Timeout
2	Drive in fault. Reset with MM_MAFR, then enable again with MM_MSO
3	Homing error reported by the drive (MM_MAH only)



When you mix the two families on one axis, for example MM_Profile_Torque_dfl in a Rockwell-like project, drop the *Start* of the DS402 block before a Rockwell-like command takes over: the DS402 Profile blocks hold their mode as long as *Start* is true.

5.2. Ladder or Structured Text

Install the **ladder** set: it opens on any Studio 5000 seat. The seven command blocks also exist in Structured Text, with an `_ST` suffix, the same pins and the same behaviour, for those who maintain the code. MM_MAUI, MM_MAUO and MM_MSL are ladder only.



Structured Text needs a licence in Studio 5000. On a seat without it an `_ST` block cannot be opened or downloaded: ship the ladder set to customers who edit their own program.

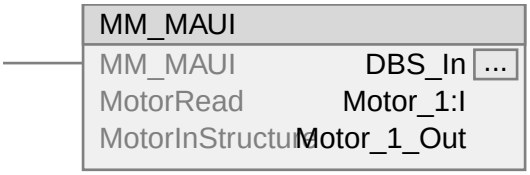
5.3. Equivalence with the DS402 AOI

Rockwell-like	DS402 equivalent	On the drive
MM_MAUI	MM_ReadData_dfl	Input assembly `Motor_x:I` into `DBS55_Out`
MM_MAUO	MM_SendData_dfl	`DBS55_In` into output assembly `Motor_x:O`
MM_MSO	*TorqueEnable* / *Start* of the Profile blocks	Up to *Operation Enabled*, axis holding position
MM_MSF	Dropping *TorqueEnable* / *Start*	Power stage off
MM_MAFR	MM_Reset_Error_dfl	Fault reset
MM_MAM	MM_Profile_Position_dfl	Mode 1, Profile Position
MM_MAJ	MM_Profile_Velocity_dfl	Mode 3, Profile Velocity
MM_MAH	MM_Profile_Homing_dfl	Mode 6, Homing
MM_MAS	*Halt* bit of the control word	Stop and hold in place
MM_MSL	Position limit pins of the Profile blocks	Software position limits, 607Dh
MM_Status_dfl	MM_Status_dfl	Same block for both families

The DS402 blocks are modes, kept alive by *Start*; the Rockwell-like blocks are commands, fired on an edge and closed by *Done*. There is no Rockwell-like Profile Torque: use MM_Profile_Torque_dfl.

5.4. MM_MAUI

Motion Axis Update Input, the first block of the scan for the axis. *MotorRead* takes the module input tag `Motor_x:I`, *MotorInStructure* the `DBS55_Out` that every other block reads.

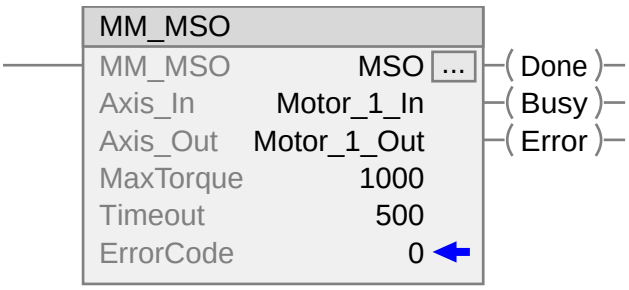


5.5. MM_MSO

Motion Servo On. Brings the drive to *Operation Enabled*: the motor holds the position it is in. It is the first command to give to an axis.

Pin	Meaning	Default
MaxTorque	Torque limit while enabled, thousandths of rated	1000
Timeout	ms allowed for the sequence	5000

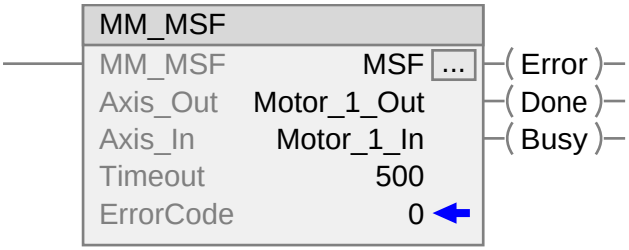
While enabling, the block writes mode 3 with a target speed of 0, so *Operation Enabled* is a standstill. MM_MAM, MM_MAJ and MM_MAH then set their own mode. With the drive in fault the block ends at once with *ErrorCode* = 2.



5.6. MM_MSF

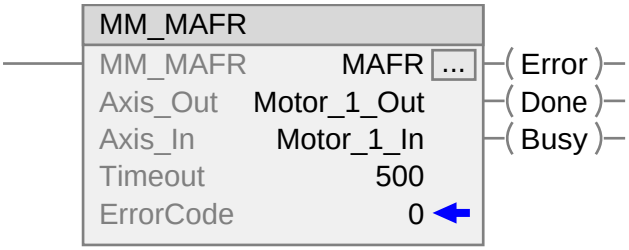
Motion Servo Off. Switches the power stage off: the axis is no longer held. It completes with *Done* also when the drive is already off or in fault, so it is safe to call on any axis.

MM_MSF is not an emergency stop. Safety functions must never rely on a fieldbus command.



5.7. MM_MAFR

Motion Axis Fault Reset. Resets a drive fault with the CiA 402 edge on control word bit 7. Without a fault it completes at once with *Done*, so it can go in a general reset. After the reset the drive is disabled: give MM_MSO again.



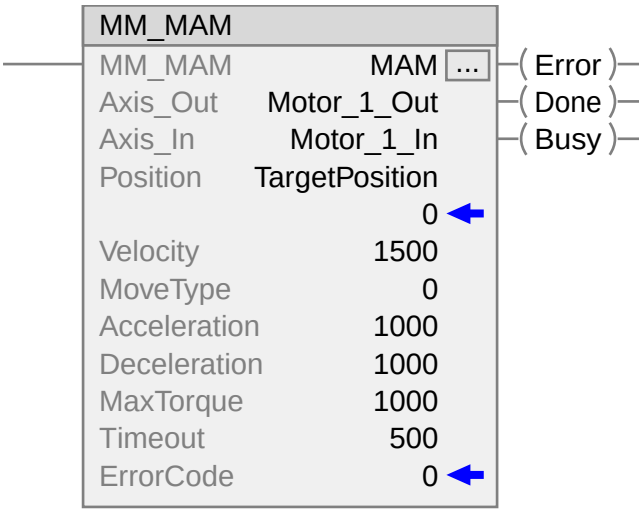
5.8. MM_MAM

Motion Axis Move. Trapezoidal positioning in Profile Position. *Done* comes up when the target has been reached.

Pin	Meaning	Default
Position	Target, pulses or scaled units	0
MoveType	0 absolute, 1 incremental, as on a Logix MAM	0
Velocity	rpm	0
Acceleration, Deceleration	rpm/s	0
MaxTorque	Torque limit, thousandths of rated	1000
Timeout	ms for the set-point handshake. The travel is not timed	5000



In revision 1.0 *MoveType* was called *Direction* and had the opposite polarity. See Updating the Rockwell-Like AOI from Revision 1.0 before updating a running machine.

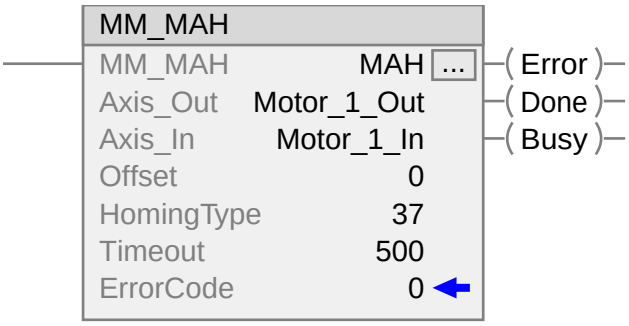


5.9. MM_MAH

Motion Axis Home. Runs the homing of the drive. Method 37, the default, is homing in place: the current position becomes the zero, without moving.

Pin	Meaning	Default
HomingType	CiA 402 homing method, see the DBS/FC manual	37
Offset	Position assigned at the end of the homing	0
Timeout	ms for the whole homing. Size it on the slowest search	5000

A homing error reported by the drive ends the block with *ErrorCode* = 3.



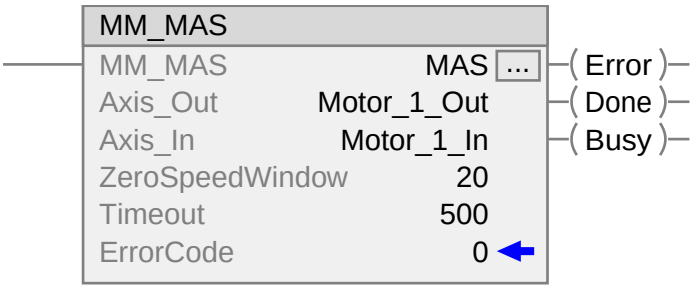
5.10. MM_MAS

Motion Axis Stop. Stops the movement in progress, a jog or a positioning, on the deceleration ramp, then holds the axis where it stopped, enabled and ready for the next command.

Pin	Meaning	Default
ZeroSpeedWindow	Standstill window, ± rpm. Widen it on high inertia axes or with large gear ratios	20
Timeout	ms allowed after the standstill to engage the hold	5000

Drop the rung of the MM_MAJ that started the jog before calling MM_MAS. An MM_MAJ still true rewrites the speed, and the jog starts again.

The axis is left in Profile Position, mode 1, which is what holds it. Keep it in mind if a DS402 Profile block comes next.



5.11. MM_MAJ

Motion Axis Jog. Turns the motor in Profile Velocity until MM_MAS stops it.

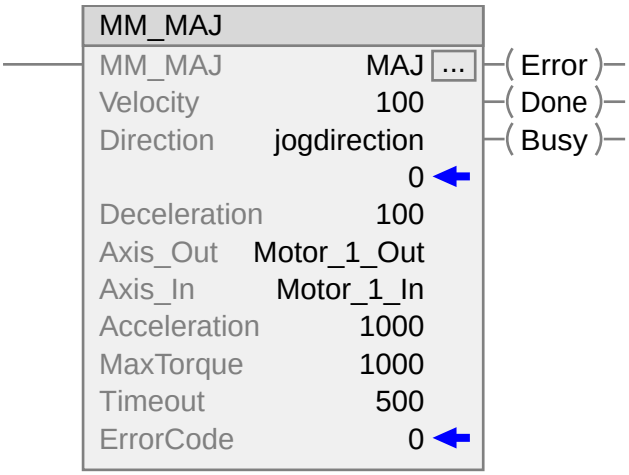
Pin	Meaning	Default
Direction	1 positive sense of the drive, 0 the other	0
Velocity	rpm, taken in absolute value	0
Acceleration, Deceleration	rpm/s	0
MaxTorque	Torque limit, thousandths of rated	1000

Pin	Meaning	Default
Timeout	ms for the start of the jog only	5000

While the rung is true the speed is rewritten at every scan, so *Velocity* can be changed on the fly. When the rung drops the motor keeps turning at the last speed: only MM_MAS, MM_MSF or a fault stop it. *Done* here means *command issued*.



In revision 1.0 dropping the rung stopped the jog. See Updating the Rockwell-Like AOI from Revision 1.0 before updating a running machine.



5.12. MM_MSL

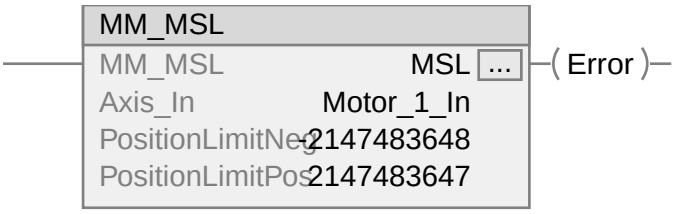
Motion Set Limits. Writes the software position limits, object 607Dh, into `DBS55_In`. The drive applies the limits it receives with the process data, not P164 and P166, and a new `DBS55_In` holds 0 and 0, which keeps the axis still: one MM_MSL per axis solves it.

Pin	Meaning	Default
PositionLimitNeg	Negative limit, same unit as MM_MAM	-2147483648
PositionLimitPos	Positive limit	2147483647
Error	*PositionLimitPos* not greater than *PositionLimitNeg*: nothing is written	

Put it on an always true rung, before MM_MAUO. The defaults disable the limits, so an instance left as it is already gives a working axis.

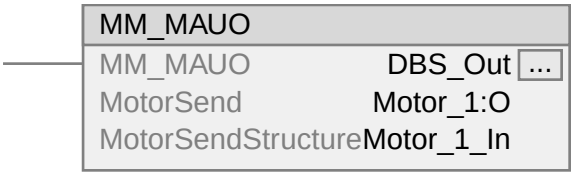


MM_Profile_Position_dfl and MM_Profile_Velocity_dfl write the limits themselves: on an axis driven by those blocks MM_MSL is not needed.



5.13. MM_MAUO

Motion Axis Update Output, the last block of the scan for the axis. *MotorSendStructure* takes the `DBS55_In` written by the command blocks, *MotorSend* the module output tag `Motor_x:O`.



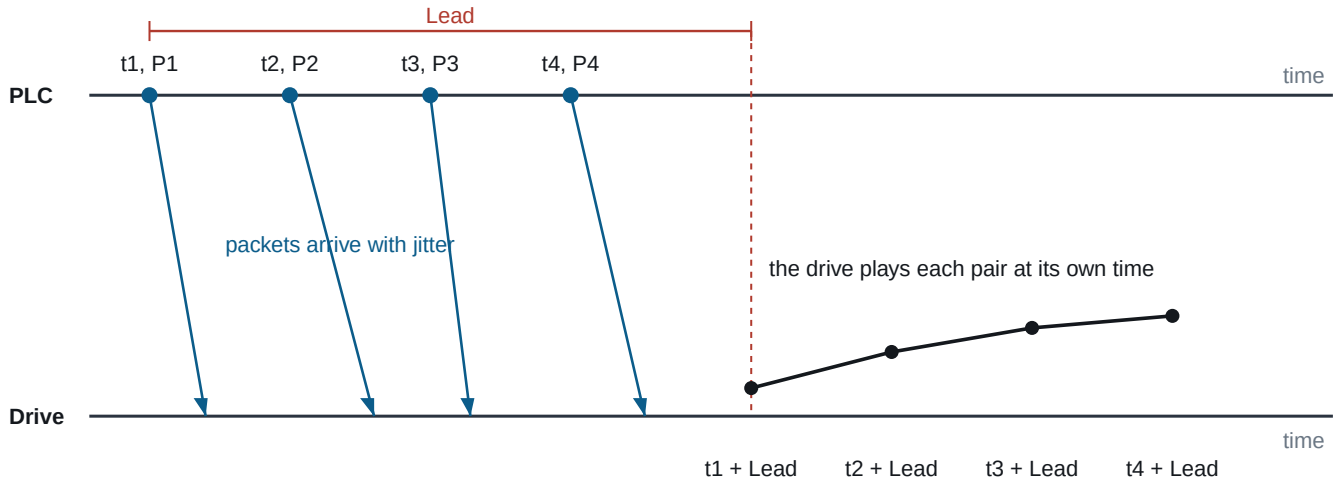
6. CIP Sync AOI

Here the trajectory is computed by the motion planner of the Logix controller, on a virtual axis, and the drive follows it in cyclic synchronous position. Whatever you do with a Logix axis, a MAJ, a MAM, a gear or a cam, the drive reproduces on the motor shaft.

Download: AOI and Program Example (CIP Sync).

6.1. How It Works

EtherNet/IP has no fixed bus cycle: the task, the RPI and the packets all jitter a little, and that jitter would end up as noise on the motor speed. CIP Sync moves the timing into the data. Controller and drive share one clock through PTP (IEEE 1588), and with every target position the PLC sends the instant it refers to, *SetpointTime*. The drive keeps the last three pairs and plays them on their own time stamps, so a packet that arrives early, late or not at all does not change the trajectory.



The time stamp is set ahead of the current time by a *lead*, which must cover task, RPI and network: the motor follows the virtual axis delayed by that lead. In the other direction the drive stamps its actual values with *PositionTime*.

6.2. Requirements

- **CSP option** on the drive: the same one that unlocks cyclic synchronous position on EtherCAT and POWERLINK.
- NetX90 fieldbus board, firmware 5.005 or later.
- P176 = 5 and `DBS55_CIP_sync.eds`.
- A Logix controller with CIP Sync time synchronization and a motion group with virtual axes. The drive stays a generic EtherNet/IP device: no axis of the drive is created in the motion group.

6.3. Setting Up the Project

Addresses, EDS and module are added as in the previous chapters, with the CIP Sync EDS. Then:

Where	Setting
Controller Properties > Date/Time	Tick *Enable Time Synchronization*. Without another grandmaster on the network the controller becomes the master, which is fine. Set date and time before starting: changing them later makes the whole network jump
Drive module	RPI of the axis, 2 ms in the example. The connection carries 40 bytes in and 30 out
Motion group	*Coarse Update Period* equal to the RPI
Virtual axis	`AXIS_VIRTUAL` in the motion group, **Linear**, units of your choice (revolutions in the example). Set maximum speed, acceleration and deceleration: with the defaults of a new axis the first MAJ fails
Event task	Trigger *Motion Group Execution* on the motion group, high priority. It runs the MM_CS blocks; the motion instructions stay in the continuous task

Once online, the *Time Sync* tab of the drive module shows its clock *Synchronized*, slave on port 1, a few tens of nanoseconds from the master. A virtual axis needs no MSO.

6.4. Data Types

The blocks carry three data types. `DBS55S_In` and `DBS55S_Out` mirror the two assemblies byte by byte, so one copy moves a whole frame and time and values always belong to the same sample.

Byte	`DBS55S_In` member	Type	Meaning
0	PositionTime	LINT	Object 3071h. Time of the actual values, ns of the PTP clock
8	StatusWord	INT	CiA 402 status word
10	ModeOfOpDisplay	SINT	Mode of operation display
12	ActualPos	DINT	Actual position, pulses
16	ActualVel	DINT	Actual speed, rpm
20	FollowingError	DINT	Following error, pulses
24	ActualTrq	INT	Actual torque, thousandths of rated
26	ActualAlarm	INT	Current alarm, 0 = none
28	DcVoltage	INT	DC link voltage, 0.1 V
30	Din	INT	Digital inputs
32	SyncStatus	INT	Object 3072h. Bit 0 PTP synchronized, 1 time valid, 2 sync0 ok, 3 locked, 4 set point time received

Byte	`DBS55S_Out` member	Type	Meaning
0	SetpointTime	LINT	Object 3070h. Time the target refers to, ns of the PTP clock
8	ControlWord	INT	CiA 402 control word
10	ModeOfOp	SINT	8 to follow, 6 for homing
12	TargetPos	DINT	Target position, pulses
16	VelocityOffset	DINT	Velocity feed-forward, rpm

Byte	DBS55S_Out member	Type	Meaning
20	HomingOffset	DINT	Position assigned at the end of homing, pulses
24	HomingMethod	SINT	CiA 402 homing method

MM_CS_Time holds the time of the task: one tag per task, written by MM_CST and read by the other blocks.

6.5. The Blocks

One block per rung in the Event task, in this order. With more drives, MM_CST is called once and the rest is repeated per axis.

Order	Block	Job
1	MM_CST	PTP time of the controller and time stamp of the sample
2	MM_CSI	Input frame into DBS55S_In, time base check
3	MM_CSSTAT	Drive status, optional
4	MM_CSPWR	Enables the drive
5	MM_CSHOME	Homing, on request. After MM_CSPWR, because it overrides control word and mode while it works
6	MM_CSVA	Virtual axis position into the drive target
7	MM_CSO	Time stamp and target out, in the same frame. Always last

As in the other families, the pins not wired in the box are written on the instance, for example CS_PWR.Enable.

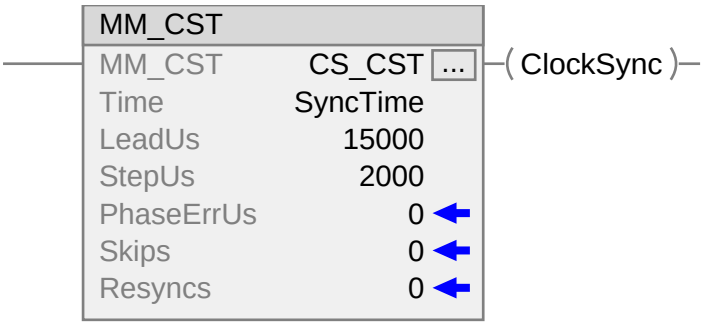
6.5.1. MM_CST

CIP Sync Time. Reads the PTP time of the controller and writes the time stamp of the sample into MM_CS_Time.

Pin	Meaning	Default
LeadUs	Lead of the time stamp over the current time, µs. See Tuning	6000
StepUs	Coarse update period, µs: the stamp advances by exactly one period per execution, on the planner grid. 0 = current time + lead, for a program without motion group	0
ClockSync	1 when the controller is a slave of another grandmaster, 0 when it is the master itself. Both are valid	
PhaseErrUs	How far the task runs from the grid. Stays within half a period	
Skips	Missed executions, caught up by whole periods	
Resyncs	Times the stamp was realigned to the current time: once at start up, then only after a clock step	



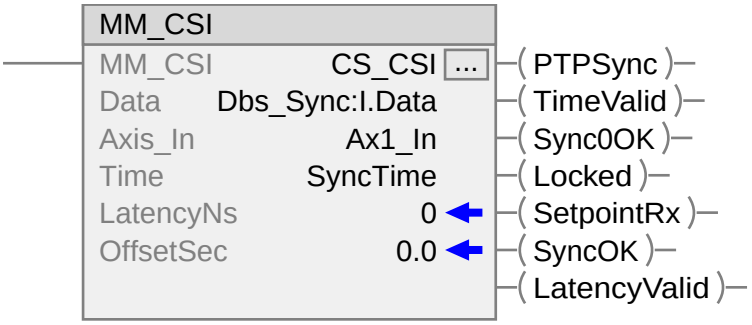
StepUs must equal the coarse update period. If it does not, *Resyncs* grows steadily and the motion is rough.



6.5.2. MM_CSI

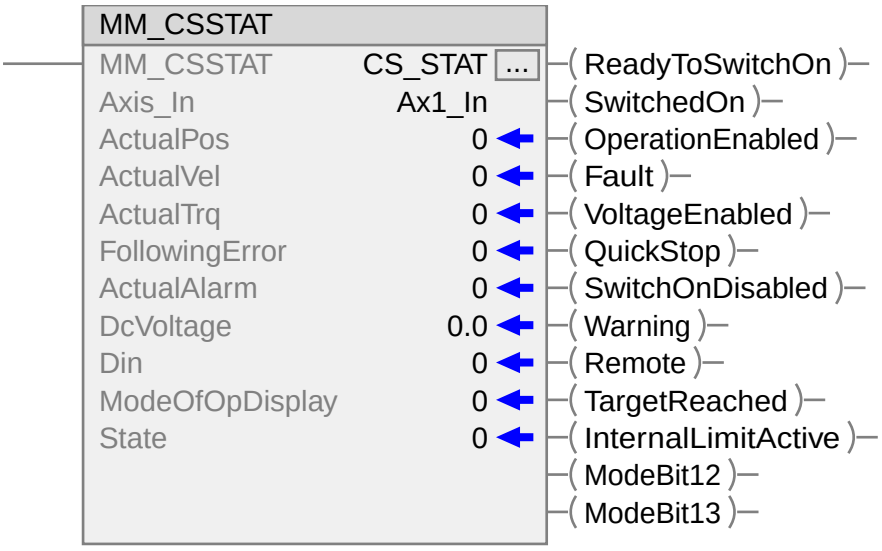
CIP Sync Input. Copies the 40 input bytes into `DBS55S_In` with one copy and decodes *SyncStatus*.

Pin	Meaning
SyncOK	Drive locked to the time base: PTP synchronized, time valid, sync0 present, locked. The one to watch
SetpointRx	The drive is receiving valid time stamps
LatencyNs	Age of the feedback when the task reads it. Positive and stable, a few ms
OffsetSec	Same difference in seconds. Close to 0 when controller and drive count the same time



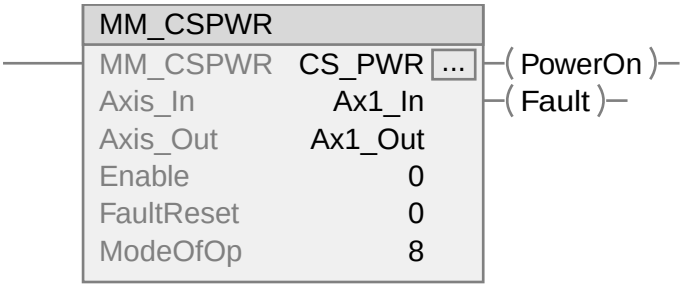
6.5.3. MM_CSSTAT

CIP Sync Status, the counterpart of MM_Status_dfl: actual values, DC voltage in volts, digital inputs, mode display, status word bits. *State* gives the CiA 402 state as a number: 0 not ready to switch on, 1 switch on disabled, 2 ready to switch on, 3 switched on, 4 operation enabled, 5 quick stop active, 6 fault reaction active, 7 fault. *ModeBit12* and *ModeBit13* are *Following set point* and *Following error* in mode 8, *Homing attained* and *Homing error* in mode 6.



6.5.4. MM_CSPWR

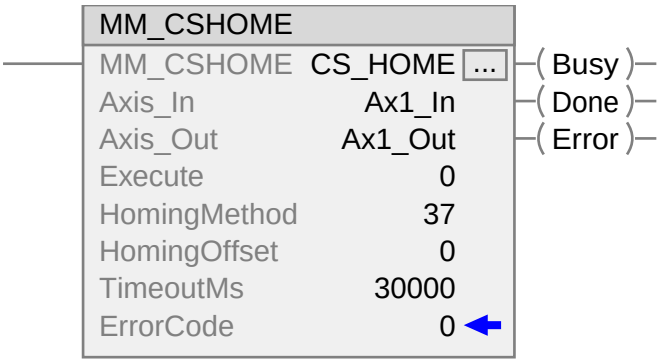
CIP Sync Power. A level, not a command: *Enable* at 1 brings the drive to *Operation Enabled* and keeps it there, at 0 switches the power stage off. *ModeOfOp* is the mode written to the drive, 8. In fault the control word is 0 and *FaultReset* raises bit 7, one rising edge per reset. *PowerOn* is the state of the drive.



6.5.5. MM_CSHOME

CIP Sync Homing. Rising edge of *Execute*, with the drive enabled: mode 6, homing with the same handshake as MM_MAH. Unlike MM_MAH, the drive **stays in mode 6** while *Execute* is high: that window is where the virtual axis is aligned with the motor, see Example Program. When *Execute* drops, the drive goes back to mode 8.

Pin	Meaning	Default
HomingMethod	CiA 402 method, 37 homing in place	37
HomingOffset	Position assigned at the end, pulses	0
TimeoutMs	ms for the whole homing	30000
ErrorCode	1 timeout, 2 fault, 3 homing error, 4 drive not enabled	



6.5.6. MM_CSVA

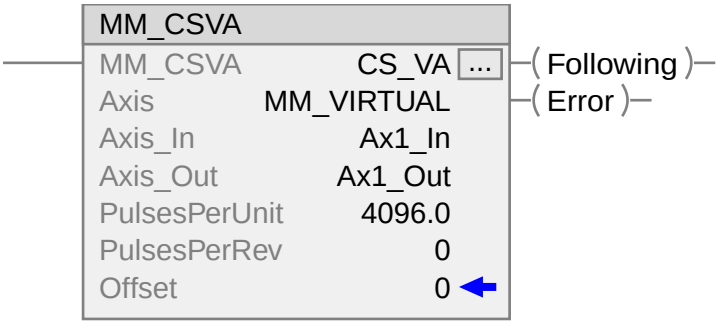
CIP Sync Virtual Axis. *TargetPos* = *CommandPosition* of the virtual axis × *PulsesPerUnit* + offset. The offset follows the actual position until the drive is in *Operation Enabled* and mode 8; from then *Following* is high and the offset is frozen, so enabling or coming back from a homing never makes the axis jump.

Pin	Meaning	Default
PulsesPerUnit	Encoder pulses per unit of the virtual axis: pulses per revolution with the axis in revolutions	1.0
PulsesPerRev	Pulses per motor revolution, to send the velocity feed-forward. 0 = none	0
Error	Target beyond ±2·10⁹ pulses, held	

The axis position is a REAL: above about 16 million pulses (4000 revolutions at 4096 pulses) the single pulse is lost. A machine that keeps turning one way should redefine the virtual axis with MRP before getting there.

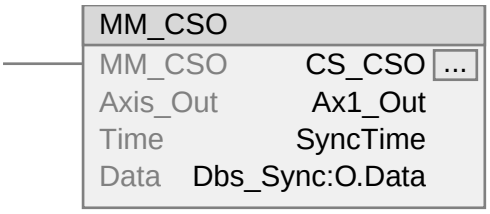


Never MRP the virtual axis while *Following* is high: the motor follows the jump. Do it with the drive disabled or in the homing window.



6.5.7. MM_CSO

CIP Sync Output. Writes the time stamp into `DBS55S_Out` and sends the first 30 bytes with one copy: time and target always travel in the same packet.



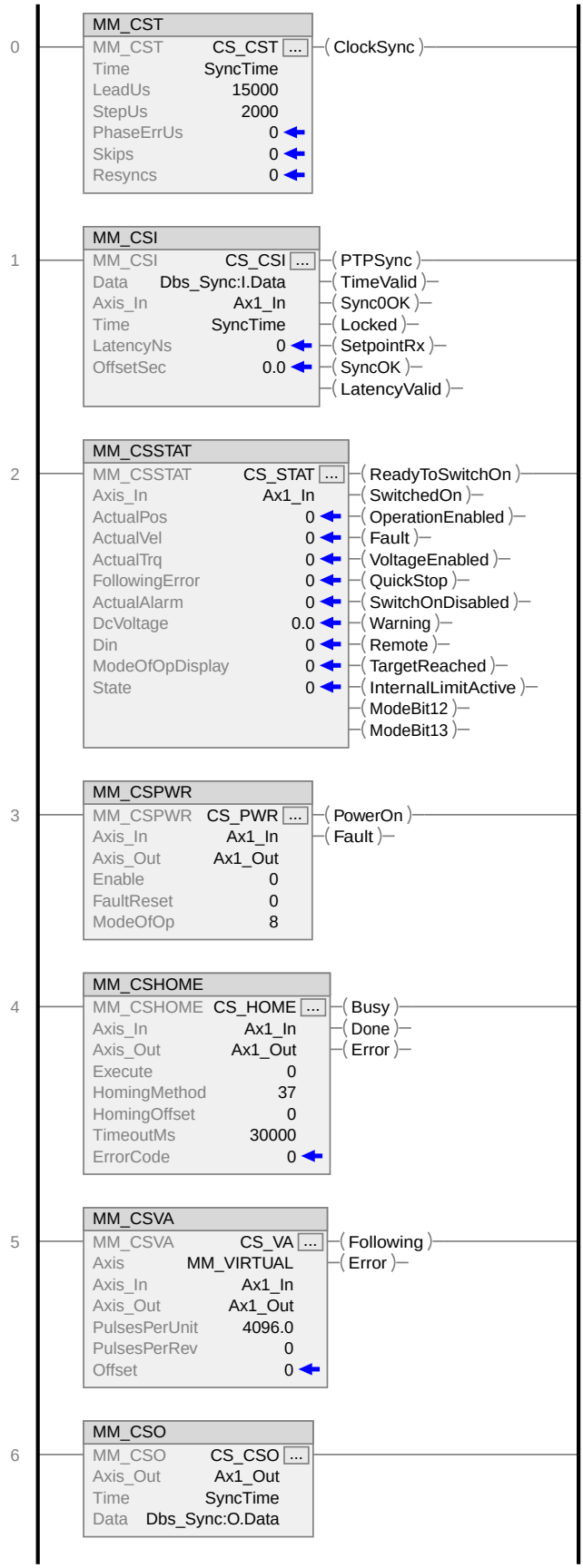
6.6. Example Program

One drive follows one virtual axis, with jog, move, stop and a homing that aligns the virtual axis with the motor.

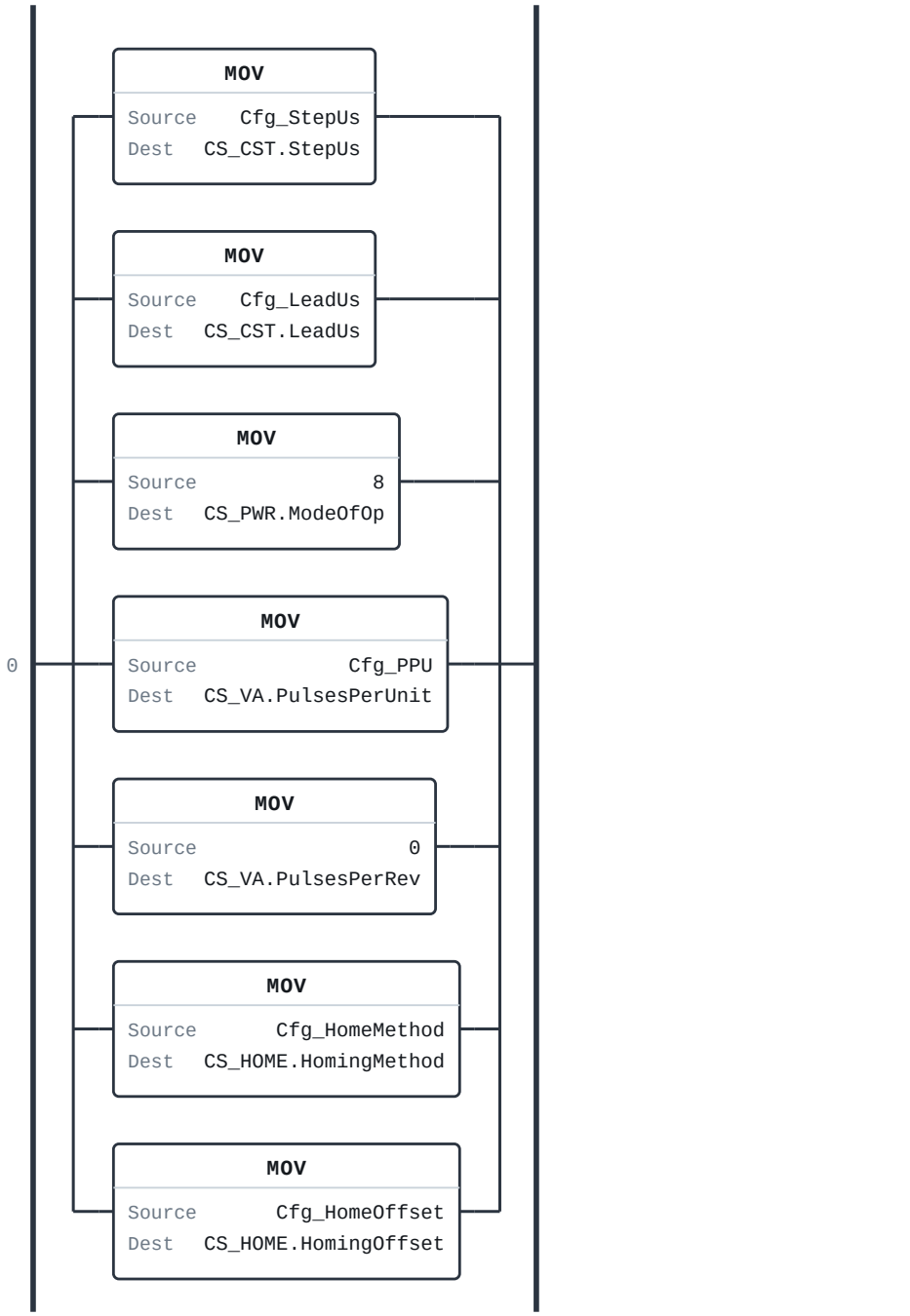
Item	Setting
Drive module `Dbs_Sync`	From `DBS55_CIP_sync.eds`, RPI 2 ms
Motion group `MM_SYNC`	Coarse update period 2 ms
Virtual axis `MM_VIRTUAL`	Linear, revolutions, maximum speed 10 rev/s, acceleration and deceleration 100 rev/s²
Event task	*Motion Group Execution* on `MM_SYNC`, routine Sync

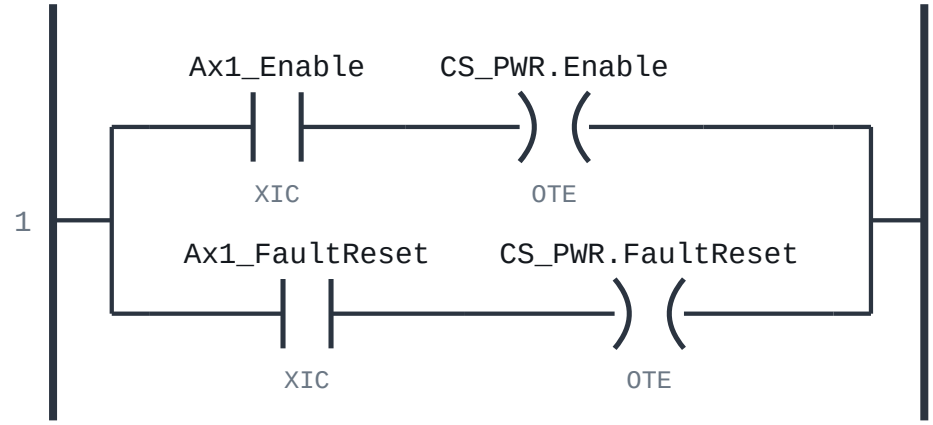
Tag	Type	Use
`CS_CST` … `CS_CSO`	The block types	Instances
`SyncTime`	`MM_CS_Time`	Time of the Event task
`Ax1_In`, `Ax1_Out`	`DBS55S_In`, `DBS55S_Out`	Data of the drive
`Cfg_StepUs`, `Cfg_LeadUs`	DINT	2000 and 15000
`Cfg_PPU`	REAL	4096.0
`Cfg_HomeMethod`, `Cfg_HomeOffset`	SINT, DINT	37 and 0
`Ax1_Enable`, `Ax1_FaultReset`	BOOL	Enable, fault reset
`Cmd_Jog`, `Cmd_Move`, `Cmd_Stop`, `Cmd_Home`	BOOL	Commands, rising edge
`Jog_Dir`, `Jog_Speed`, `Jog_Acc`, `Move_Dist`	DINT, REAL	Direction, rev/s, rev/s², rev
`Home_Req`, `Home_Fail`, `Home_ErrCode`, `Align_Pos`	BOOL, DINT, REAL	Homing sequence
`MAJ_Ctrl`, `MAM_Ctrl`, `MAS_Ctrl`, `MAS_Drop`, `MRP_Ctrl`	MOTION_INSTRUCTION	Motion instructions
`*_ONS`	BOOL	One shots

Routine Sync, in the Event task: the seven blocks in the order of The Blocks. `Time` goes to `SyncTime`, `Data` to `Dbs_Sync:I.Data` (MM_CSI) and `Dbs_Sync:O.Data` (MM_CSO), the axis pins to `Ax1_In`, `Ax1_Out` and `MM_VIRTUAL`.

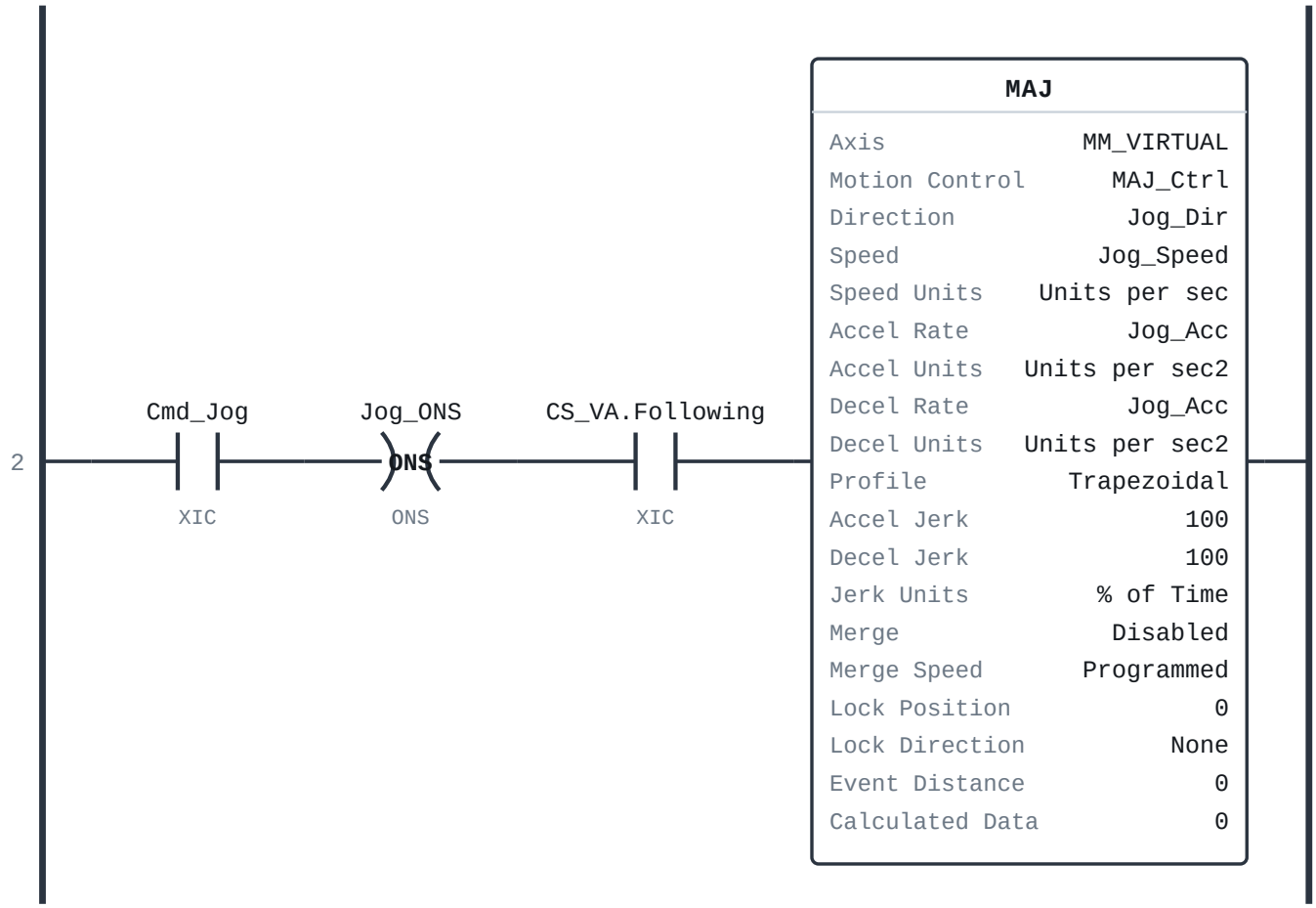


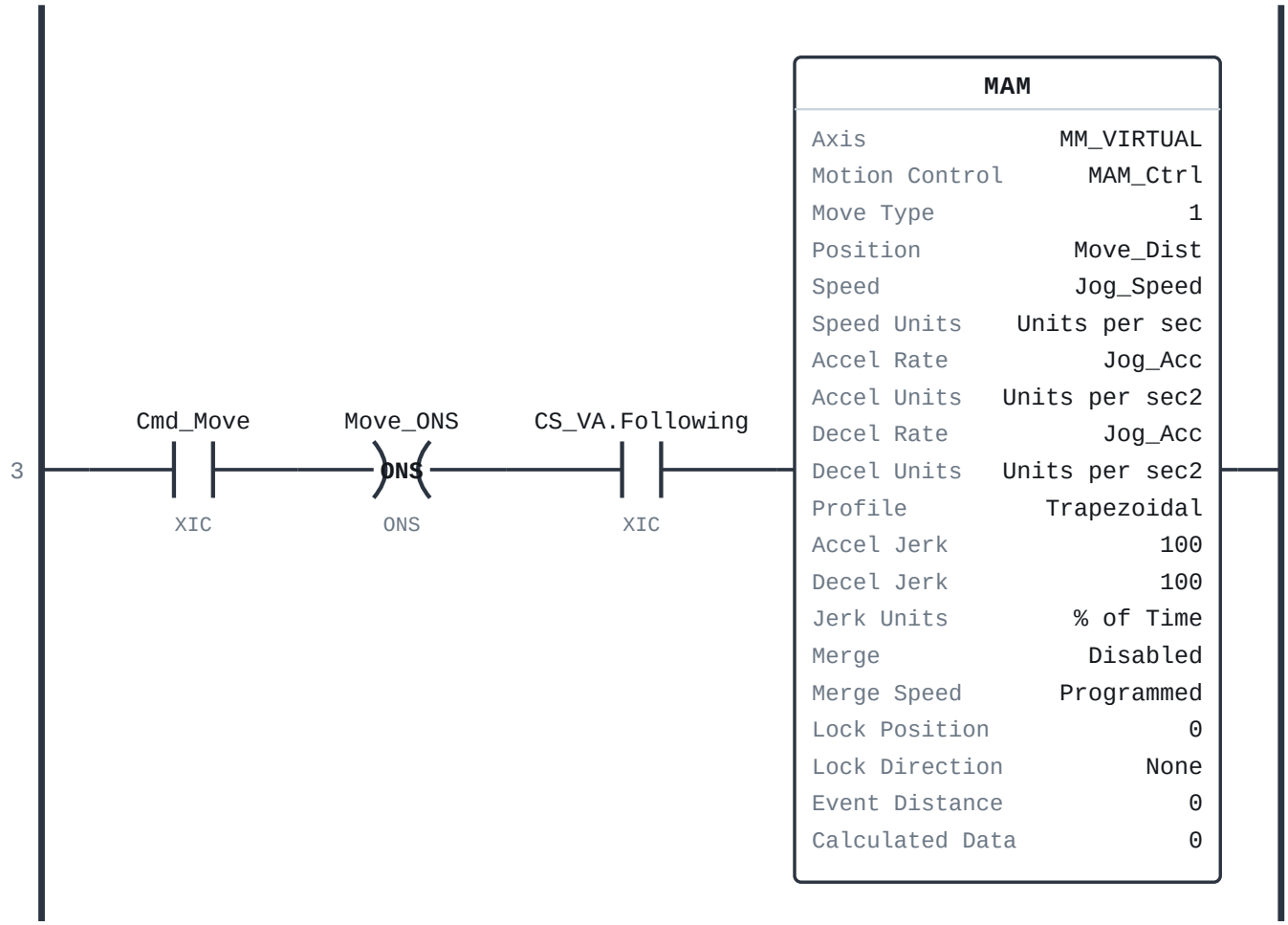
Routine Main, in the continuous task. Rung 0 writes the configuration into the instances, rung 1 passes enable and reset.

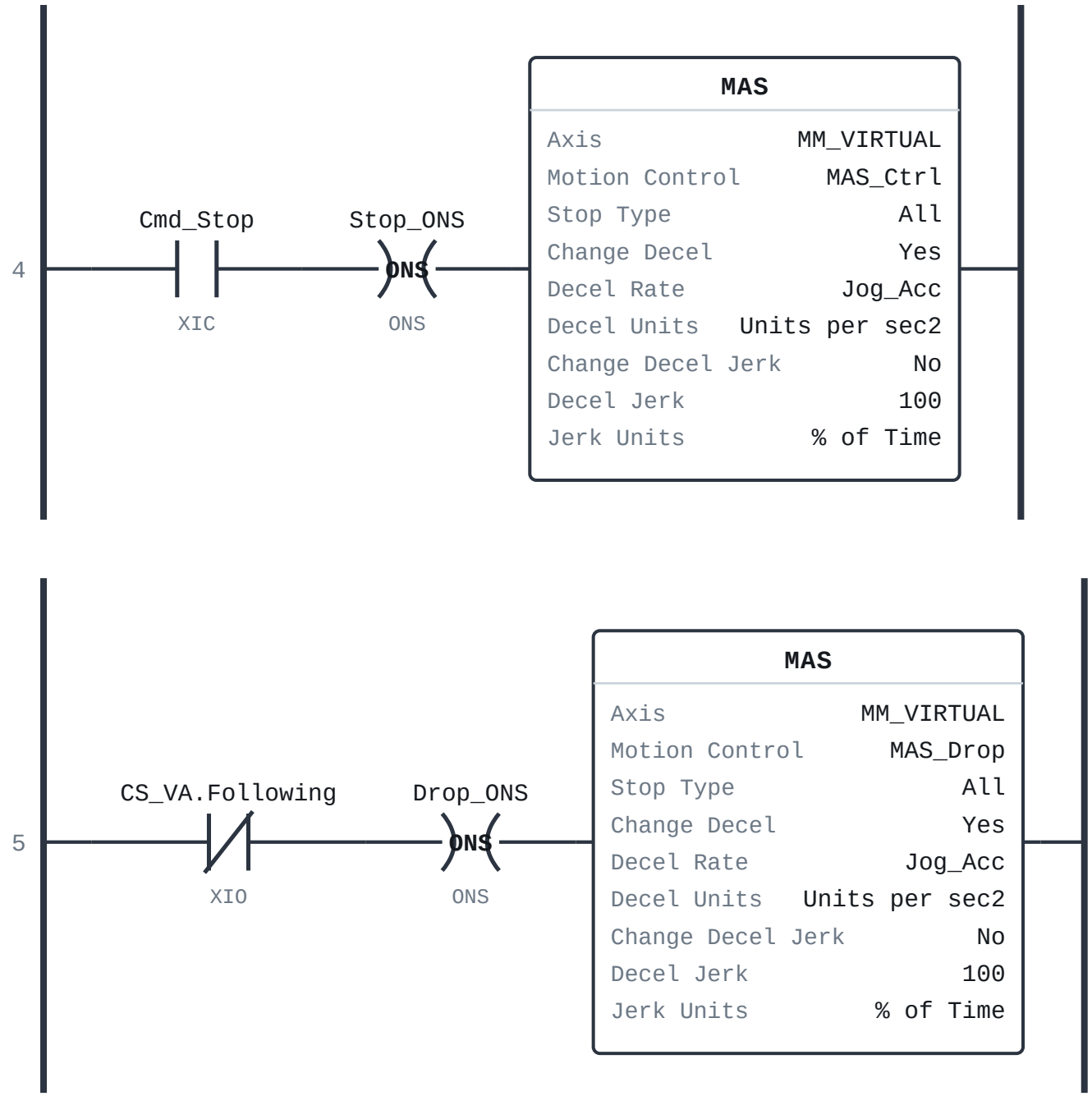




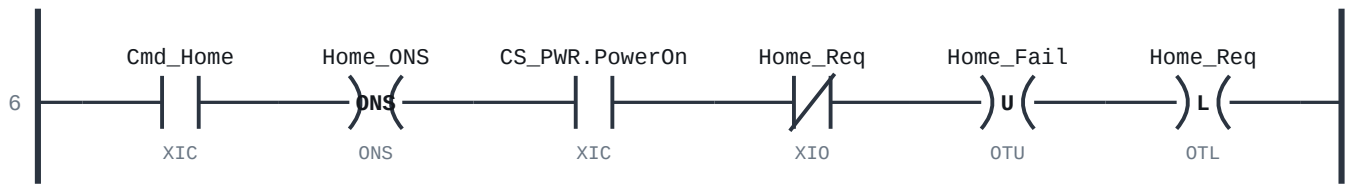
Rungs 2 to 4: jog, incremental move and stop, allowed only when `CS_VA.Following` is high. Rung 5 stops the virtual axis when the drive stops following.

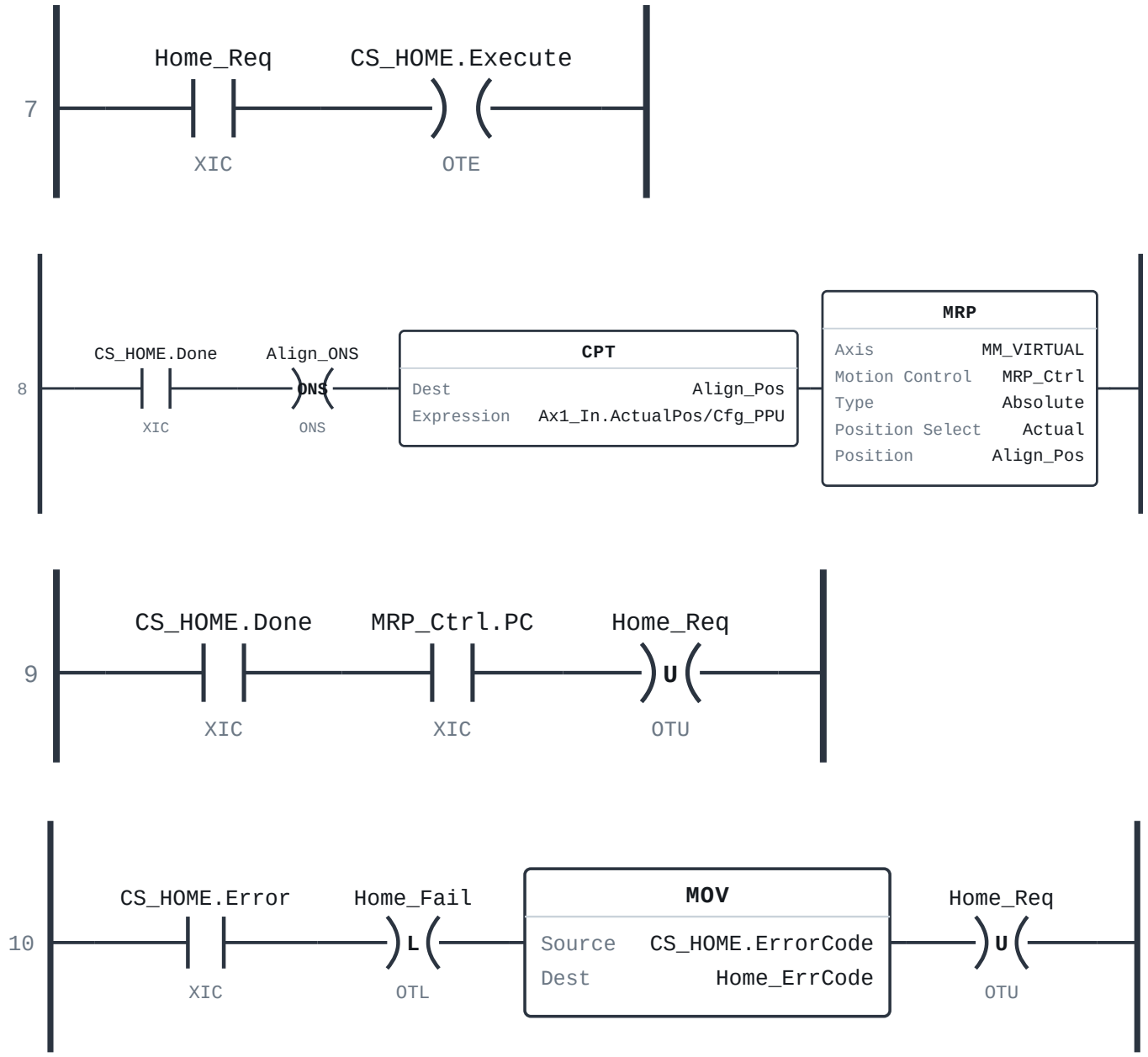

2
Cmd_Jog
XIC
Jog_ONS
ONS
ONS
CS_VA.Following
XIC
MAJ
Axis MM_VIRTUAL
Motion Control MAJ_Ctrl
Direction Jog_Dir
Speed Jog_Speed
Speed Units Units per sec
Accel Rate Jog_Acc
Accel Units Units per sec2
Decel Rate Jog_Acc
Decel Units Units per sec2
Profile Trapezoidal
Accel Jerk 100
Decel Jerk 100
Jerk Units % of Time
Merge Disabled
Merge Speed Programmed
Lock Position 0
Lock Direction None
Event Distance 0
Calculated Data 0


3
Cmd_Move
XIC
Move_ONS
ONS
ONS
CS_VA.Following
XIC
MAM
Axis MM_VIRTUAL
Motion Control MAM_Ctrl
Move Type 1
Position Move_Dist
Speed Jog_Speed
Speed Units Units per sec
Accel Rate Jog_Acc
Accel Units Units per sec2
Decel Rate Jog_Acc
Decel Units Units per sec2
Profile Trapezoidal
Accel Jerk 100
Decel Jerk 100
Jerk Units % of Time
Merge Disabled
Merge Speed Programmed
Lock Position 0
Lock Direction None
Event Distance 0
Calculated Data 0



Rungs 6 to 10: homing with alignment. `Execute` stays high for the whole sequence, so the drive stays in mode 6. When the homing is done, rung 8 sets the virtual axis to the motor position in revolutions with MRP; rung 9 then releases `Execute`, the drive returns to mode 8 and from there virtual axis and motor show the same position. Rung 10 keeps the reason of a failure.





First run.

1. `Ax1_Enable` = 1: `CS_PWR.PowerOn` and `CS_VA.Following` go high, the motor holds.
2. `Cmd_Home` with method 37: `CS_HOME.Done`, then `Home_Req` drops by itself. Drive and virtual axis now show the same position.
3. `Move_Dist` = 1.0, `Cmd_Move`: the actual position grows by 4096 pulses.
4. `Cmd_Jog`, then `Cmd_Stop`.

6.7. Tuning

Setting	Guideline
`LeadUs`	Must cover task, RPI and network; too long only adds delay. Tune it on the machine: with RPI 2 ms, 10 to 15 ms gave a smooth motion on the test bench. Start high and come down while watching the following error on a constant speed jog

Setting	Guideline
`PulsesPerRev`	0 lets the drive derive the speed from the positions; the pulses per revolution send the virtual axis speed as feed-forward. With correct time stamps both are smooth: use it for the tightest following error
Coarse update period	Equal to the RPI. Different values work, but repeat or skip samples

6.8. Diagnostics

What you see	What it means
`SyncOK` low	Drive not locked: check *Enable Time Synchronization* and the *Time Sync* tab of the module
`OffsetSec` of tens of years, `PositionTime` at 0	PTP not running on the drive, or P176 not on 5
`SetpointRx` low	No valid time stamps: MM_CST or MM_CSO not running, or `Time` on another tag
`Resyncs` growing	`StepUs` different from the coarse update period
`PhaseErrUs` near half a period	The Event task is not right after the planner: check trigger and priority
Alarm 101 CSP_SYNC	Synchronization missing for more than three cycles: check RPI, cabling and network load
General error status 16#05 on the *Time Sync* tab	The drive does not expose the Time Sync object: firmware or option without CIP Sync
First MAJ on a new virtual axis in error	Maximum speed, acceleration or deceleration not configured

6.9. Test Setup

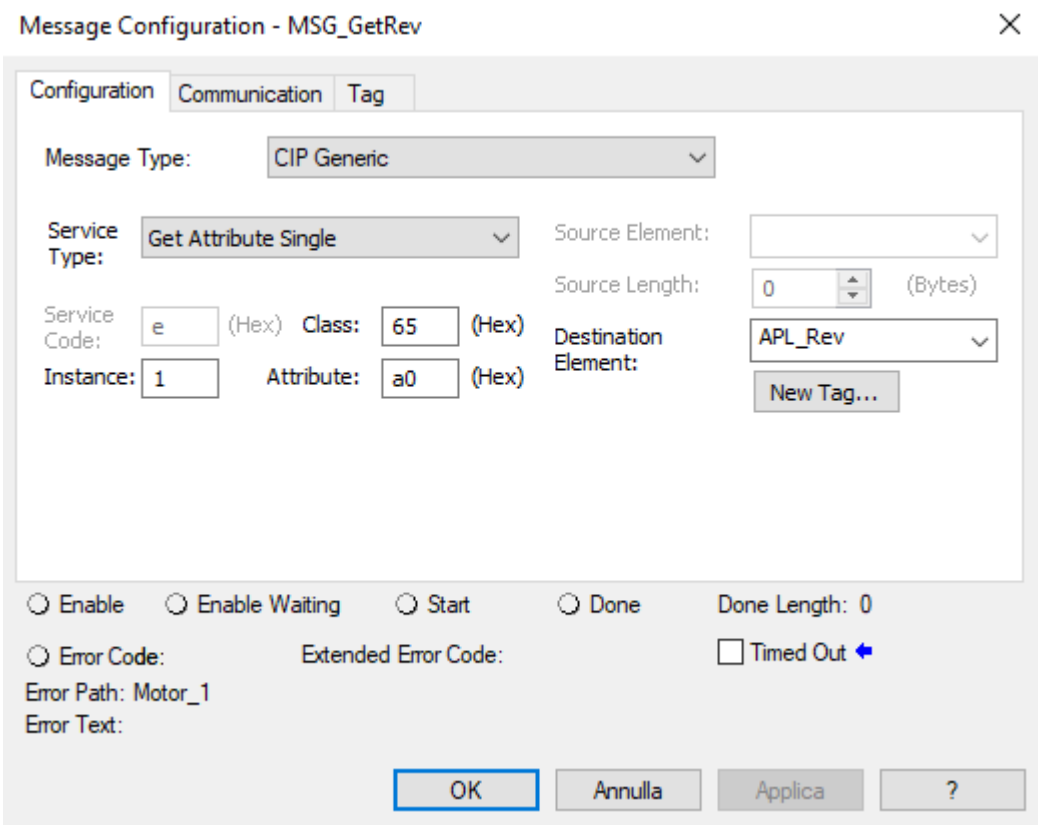
Validated on this setup, reported for reference only.

Item	Used
Controller	CompactLogix 5370 1769-L18ER-BB1B, grandmaster
Software	Studio 5000 Logix Designer 31.01
Drive	DBS with NetX90, firmware 5.005, CSP option, 4096 pulses per revolution, no load
Timing	RPI 2 ms, coarse update period 2 ms, `LeadUs` 15000

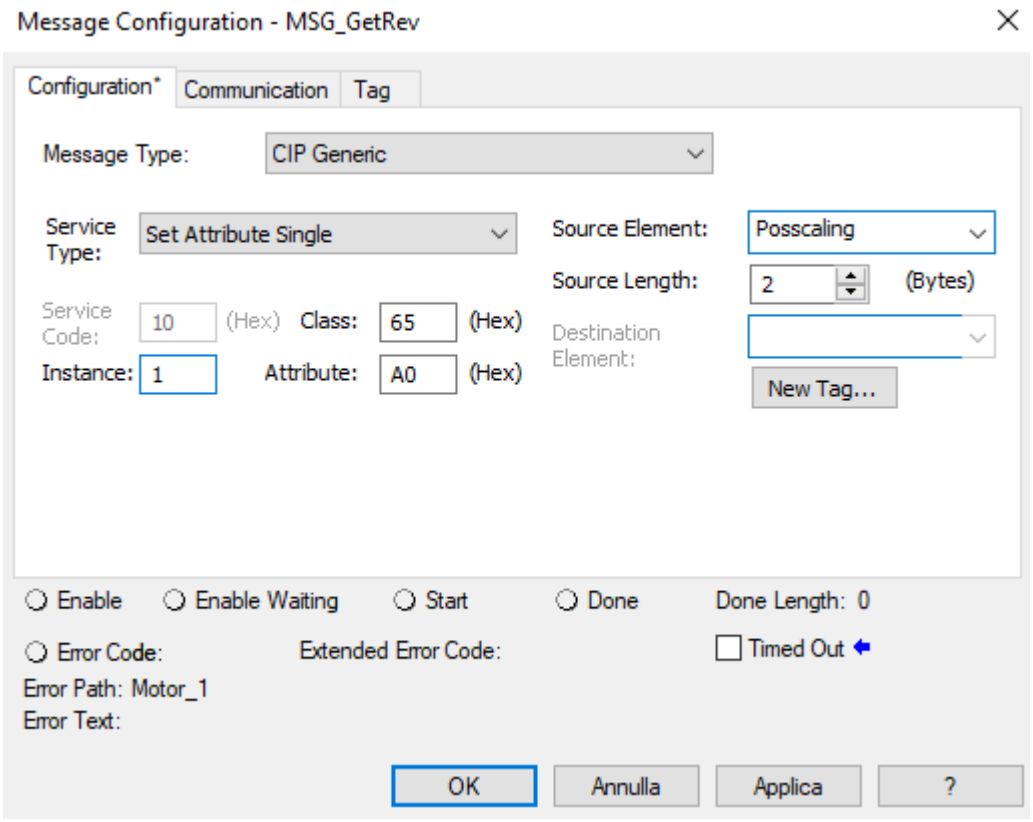
7. Writing Parameters with Async Messaging

Parameters and CiA 402 objects outside the cyclic data are read and written with Class 3 messages, Get/Set Attribute Single, on the manufacturer objects described in the EtherNet/IP chapter: COM 0x64 (information, saving), PAR 0x65 (parameters), MISC 0x66 (hardware values) and CiA 402 0x67 (object 6081h.00 is class 0x67, instance 0x60, attribute 0x81).

Example: reading P160, *PosFactorNum*: class 0x65, instance 1, attribute 0xA0, with a destination tag for the value.



To write it, the same MSG with *Set Attribute Single*. A written parameter lives in RAM until saved with the COM object.


Message Configuration - MSG_GetRev
Configuration*
Communication
Tag
Message Type:
CIP Generic
Service Type:
Set Attribute Single
Source Element:
Posscaling
Source Length:
2
(Bytes)
Service Code:
10
(Hex)
Class:
65
(Hex)
Destination Element:
Instance:
1
Attribute:
A0
(Hex)
New Tag...
Enable
Enable Waiting
Start
Done
Done Length: 0
Error Code:
Extended Error Code:
Timed Out
Error Path: Motor_1
Error Text:
OK
Annulla
Applica
?

8. Micro850 Application Example

A Micro800 controller has no EDS catalogue and runs no Add-On Instructions, but the newer CPUs can drive the motor cyclically. This chapter shows how, with Connected Components Workbench (CCW).

8.1. Which CPU

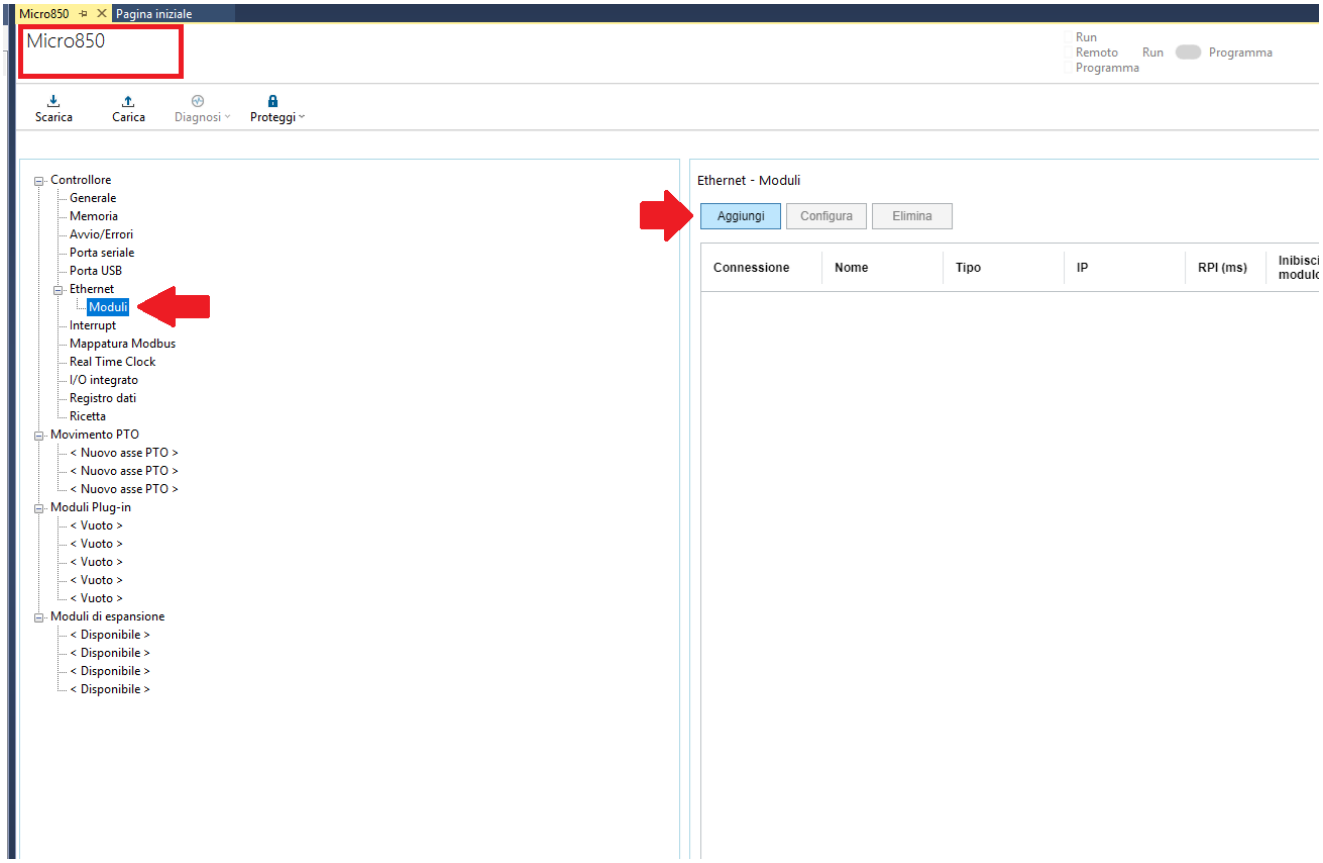
Item	Requirement
Controller	Micro850 **2080-L50E** or Micro870 **2080-L70E**, firmware 21.011 or later
Software	CCW 21 or later, Standard or Developer Edition
Drive	DBS/FC with EtherNet/IP, firmware 5.005, P176 = 0 (Default layout)
Network	Drive and controller in the same subnet, static addresses



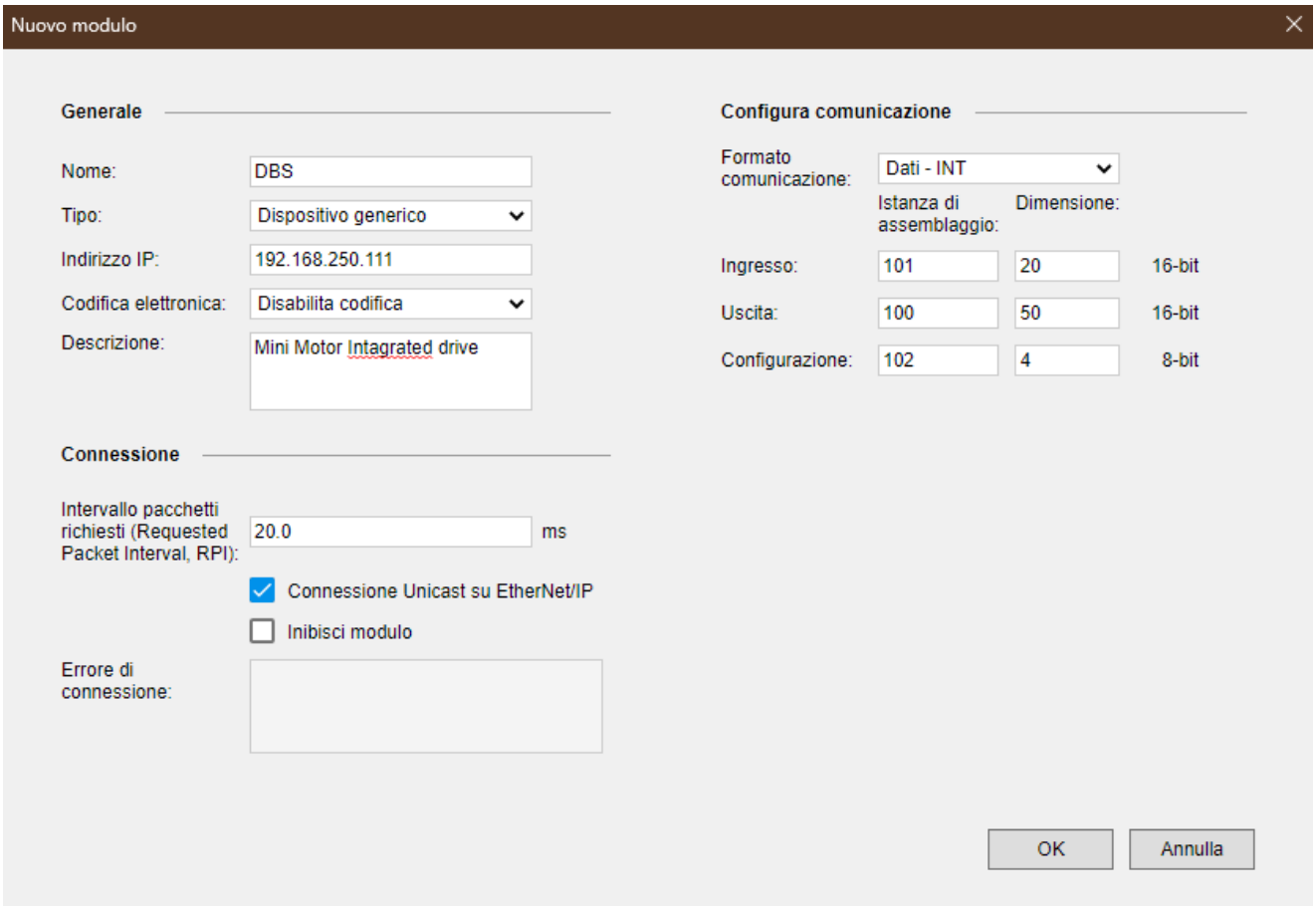
Only the catalogue numbers with the letter **E** are EtherNet/IP scanners. A 2080-LC50-xxxxx only does explicit messaging: with it the drive can only be reached as described in Reading and Writing Parameters from CCW. Check the CPU label before committing to a delivery.

8.2. Add the DBS as a Generic Device

1. Set the drive address as in Configure the Motor Address, and give the controller a static address in the same subnet.
2. In the project tree open *Controller* → *Ethernet* → *Modules* and click *Add*.



3. Select *Generic Device*, name it (`DBS` in this example), enter the drive address and set *Electronic Keying* to *Disable Keying*.
4. Fill in the communication configuration below and confirm with OK.



Field	Value	Note
Communication Format	Data - INT	The DFL layout is word aligned
Input — Assembly Instance / Size	101 / 20	Assembly 0x65, drive outputs
Output — Assembly Instance / Size	100 / 50	Assembly 0x64, drive inputs
Configuration — Assembly Instance / Size	102 / 4	Always zero
RPI	20 ms	The position loop runs in the drive: a lower RPI only loads the CPU



The sizes are those of the Default layout, P176 = 0. With any other layout the connection is refused.

CCW creates `DBS_O` (INT[0..24], to the drive), `DBS_I` (INT[0..9], from the drive) and `DBS_C` (SINT[0..3], leave at zero). Download and go online: the module must read *Running*, otherwise see Commissioning and Diagnostics.

8.3. Process Data Seen From CCW

The assemblies are arrays of 16-bit words. 32-bit values take two words, least significant first.

DBS_O word	Name	Type	Units
0	ControlWord	UINT	CiA 402 control word
1…2	TargetPosition	DINT	position units
3	TorqueLimit	UINT	thousandths of rated torque
4…5	TargetVelocity	DINT	rpm
6	TargetTorque	INT	thousandths of rated torque
7	ModeOfOperation	SINT	low byte; high byte 0
8…9	ProfileVelocity	UDINT	rpm
10…11	ProfileAccel	UDINT	rpm/s
12…13	ProfileDecel	UDINT	rpm/s
14…15	PositionLimit.Neg	DINT	position units
16…17	PositionLimit.Pos	DINT	position units
18…19	HomingOffset	DINT	position units
20	HomingMethod	SINT	low byte; high byte 0
21…24	—	—	Not used, 0

DBS_I word	Name	Type	Units
0	StatusWord	UINT	CiA 402 status word
1…2	PositionActualValue	DINT	position units
3	TorqueActualValue	INT	thousandths of rated torque
4…5	VelocityActualValue	DINT	rpm
6	VDclink	UINT	0.1 V
7	ModeOfOperation display	SINT	low byte
8	ActualAlarm	UINT	0 = no alarm
9	—	—	Padding



Check the word order online before writing logic: write 16#00010000 into the target position and verify that `DBS_O[1]` reads 0 and `DBS_O[2]` reads 1. If they are swapped, set *Swap* to TRUE in the COP calls.

8.4. Structured Text Example

Without Add-On Instructions the CiA 402 state machine is written in the program. The code below drives one axis (enable, homing, positioning, fault reset) and is meant to become a UDFB, one instance per motor. Two notes on the Micro800 ST dialect: bitwise operations on integers use `AND_MASK`, `OR_MASK` and friends, and `COP` is a function block, one instance per call, whose *Length* counts **destination** elements.

Name	Type	Description
iSw	INT	StatusWord as read
iActPos, iActVel	DINT	Actual position and velocity
iActTrq, iAlarm, iDcVolt, iModeDisp	INT	Torque, alarm, DC link voltage, active mode
oCw	INT	ControlWord to be written
oTargetPos, oTargetVel, oNegLimit, oPosLimit, oHomeOffs	DINT	Set points and limits
oProfVel, oProfAcc, oProfDec	UDINT	Profile parameters
oTrqLimit, oTargetTrq, oMode, oHomeMethod	INT	Torque limit and target, requested mode, homing method
stReady, stSwitchedOn, stOpEnabled, stFault, stSwOnDisabled, stQuickStop	BOOL	Decoded states
stTargetReached, stSetpointAck, stHomingDone, stHomingError	BOOL	StatusWord bits 10, 12 and 13
cmdEnable, cmdReset, cmdHome, cmdMove, cmdStop	BOOL	Commands from the application or the HMI
seq	INT	Sequence step
cpRdPos … cpWrHo	COP	One COP instance per copy line

8.4.1. Block 1—Reading and Decoding the Inputs

```
(* ASSEMBLY 101 -> typed variables. 32 bit items are copied with COP:
   Length counts DESTINATION elements. *)

iSw       := DBS_I[0];                              (* StatusWord        *)
iActTrq   := DBS_I[3];                              (* 0.1 % of rated    *)
iDcVolt   := DBS_I[6];                              (* 0.1 V             *)
iModeDisp := DBS_I[7];                              (* padding byte is 0 *)
iAlarm    := DBS_I[8];                              (* 0 = no alarm      *)

cpRdPos ( TRUE, DBS_I, 1, iActPos, 0, 1, FALSE );     (* words 1..2 *)
cpRdVel ( TRUE, DBS_I, 4, iActVel, 0, 1, FALSE );     (* words 4..5 *)

(* ---- CiA 402 state machine decoding ---- *)
stSwOnDisabled  := ( AND_MASK( iSw, 16#004F ) = 16#0040 );
stReady         := ( AND_MASK( iSw, 16#006F ) = 16#0021 );
stSwitchedOn    := ( AND_MASK( iSw, 16#006F ) = 16#0023 );
stOpEnabled     := ( AND_MASK( iSw, 16#006F ) = 16#0027 );
stQuickStop     := ( AND_MASK( iSw, 16#006F ) = 16#0007 );
stFault         := ( AND_MASK( iSw, 16#004F ) = 16#0008 );

stTargetReached := ( AND_MASK( iSw, 16#0400 ) <> 0 );  (* bit 10 *)
stSetpointAck   := ( AND_MASK( iSw, 16#1000 ) <> 0 );  (* bit 12, mode 1 *)
stHomingDone    := ( AND_MASK( iSw, 16#1000 ) <> 0 );  (* bit 12, mode 6 *)
stHomingError   := ( AND_MASK( iSw, 16#2000 ) <> 0 );  (* bit 13, mode 6 *)
```

8.4.2. Block 2—Enable and Motion Sequence

Steps 10 to 30 are what MM_MSO does on a Logix controller, 50 to 52 MM_MAH, 60 to 63 MM_MAM. The set-point handshake of steps 61 and 62 cannot be skipped: it is the most common reason for a motor that "does not start".

```
(* seq:  0 = idle          10..30 = power up
        40 = ready         50..52 = homing
        60..63 = position  90..92 = fault handling *)

CASE seq OF

0:   (* --- Idle: no torque --- *)
     oCw := 16#0000;
     IF stFault THEN            seq := 90;
     ELSIF cmdEnable THEN       seq := 10;
     END_IF;

10:  (* --- Shutdown: towards Ready To Switch On --- *)
     oCw := 16#0006;
     IF stFault THEN            seq := 90;
     ELSIF stReady THEN         seq := 20;
     END_IF;

20:  (* --- Switch On: power applied --- *)
     oCw := 16#0007;
     IF stFault THEN            seq := 90;
     ELSIF stSwitchedOn THEN    seq := 30;
     END_IF;

30:  (* --- Enable Operation: axis under torque --- *)
     oCw := 16#000F;
     IF stFault THEN            seq := 90;
     ELSIF stOpEnabled THEN     seq := 40;
     END_IF;

40:  (* --- Ready: waiting for a command --- *)
     oCw := 16#000F;
     IF stFault THEN            seq := 90;
     ELSIF NOT cmdEnable THEN   seq := 0;
     ELSIF cmdHome THEN
        oMode := 6;             seq := 50;   (* Homing          *)
     ELSIF cmdMove THEN
        oMode := 1;             seq := 60;   (* Profile Position *)
     END_IF;

(* ------------------------ HOMING, mode 6 ------------------------ *)
50:  oCw := 16#000F;
     IF iModeDisp = 6 THEN      seq := 51;   (* mode accepted *)
     END_IF;

51:  oCw := 16#001F;                         (* bit 4 = start homing *)
     IF stHomingDone THEN       seq := 52;
     ELSIF stHomingError OR stFault THEN
                                seq := 90;
     END_IF;

52:  oCw := 16#000F;                         (* bit 4 low: handshake done *)
     cmdHome := FALSE;
     seq := 40;
```

```
(* -------------------- PROFILE POSITION, mode 1 ------------------- *)
60:  oCw := 16#000F;
     IF iModeDisp = 1 THEN      seq := 61;
     END_IF;

61:  (* oTargetPos, oProfVel, oProfAcc and oProfDec already written.
        bit 4 = new set-point, bit 5 = change set immediately,
        add 16#0040 for a RELATIVE move.                          *)
     oCw := 16#003F;
     IF stSetpointAck THEN      seq := 62;
     ELSIF stFault THEN         seq := 90;
     END_IF;

62:  oCw := 16#000F;                        (* release bit 4 *)
     IF NOT stSetpointAck THEN  seq := 63;
     END_IF;

63:  oCw := 16#000F;
     IF stTargetReached THEN
        cmdMove := FALSE;       seq := 40;
     ELSIF stFault THEN         seq := 90;
     ELSIF cmdStop THEN
        oCw := 16#010F;                     (* bit 8 = Halt *)
     END_IF;

(* ------------------------- FAULT HANDLING ---------------------- *)
90:  oCw := 16#0000;
     IF cmdReset THEN           seq := 91;
     END_IF;

91:  oCw := 16#0080;                        (* rising edge on bit 7 *)
     seq := 92;

92:  oCw := 16#0000;                        (* release bit 7 *)
     cmdReset := FALSE;
     IF stSwOnDisabled THEN     seq := 0;
     END_IF;

END_CASE;
```



After a fault reset the drive is disabled and needs a new `cmdEnable`. If the cause is still there the fault returns at once: read `iAlarm` first.

8.4.3. Block 3—Writing the Outputs

```
(* Typed variables -> ASSEMBLY 100 *)

DBS_O[0]  := oCw;                                  (* ControlWord  *)
cpWrTp ( TRUE, oTargetPos, 0, DBS_O,  1, 2, FALSE );  (* TargetPos    *)
DBS_O[3]  := oTrqLimit;                            (* TorqueLimit  *)
cpWrTv ( TRUE, oTargetVel, 0, DBS_O,  4, 2, FALSE );  (* TargetVel    *)
DBS_O[6]  := oTargetTrq;                           (* TargetTorque *)
DBS_O[7]  := oMode;                                (* ModeOfOp     *)
cpWrPv ( TRUE, oProfVel,   0, DBS_O,  8, 2, FALSE );
cpWrPa ( TRUE, oProfAcc,   0, DBS_O, 10, 2, FALSE );
cpWrPd ( TRUE, oProfDec,   0, DBS_O, 12, 2, FALSE );
```

```
cpWrNl ( TRUE, oNegLimit,  0, DBS_O, 14, 2, FALSE );
cpWrPl ( TRUE, oPosLimit,  0, DBS_O, 16, 2, FALSE );
cpWrHo ( TRUE, oHomeOffs,  0, DBS_O, 18, 2, FALSE );
DBS_O[20] := oHomeMethod;                          (* HomingMethod *)
```

8.4.4. Initial Values

```
(* Executed once, on the first scan *)
IF NOT bInitDone THEN
   oProfVel    := 3000;          (* rpm                          *)
   oProfAcc    := 3000;          (* rpm/s                        *)
   oProfDec    := 3000;          (* rpm/s                        *)
   oTrqLimit   := 1000;          (* 1000 = 100 % of rated torque *)
   oHomeMethod := 37;            (* homing in place              *)
   oHomeOffs   := 0;
   oNegLimit   := -2147483648;   (* position limits disabled     *)
   oPosLimit   :=  2147483647;
   bInitDone   := TRUE;
END_IF;
```



With both position limits at 0, or with a profile velocity, acceleration or deceleration at 0, the drive enables but does not move.

8.5. Reading and Writing Parameters from CCW

The acyclic access of the previous chapter works from CCW with the `MSG_CIPGENERIC` block, on the same four objects. Reading P160 (class 0x65, instance 1, attribute 0xA0):

```
CIPAPPCFG.Service   := 16#0E;               (* Get_Attribute_Single *)
                                            (* 16#10 to write       *)
CIPAPPCFG.Class     := 16#65;               (* PAR object           *)
CIPAPPCFG.Instance  := 1;                   (* P000...P255          *)
CIPAPPCFG.Attribute := 16#A0;               (* P160                 *)

CIPTARGETCFG.Path        := '2, 192.168.250.111';  (* port 2, drive IP *)
CIPTARGETCFG.CipConnMode := 0;                     (* unconnected      *)
CIPTARGETCFG.UcmmTimeout := 1000;                  (* ms               *)
```

One message at a time, waiting for *Done* or *Error*. A written parameter lives in RAM until saved with the COM object, class 0x64 instance 1 attribute 0x10.

On a 2080-LC50 the same block is the only access: read assembly 101 and write assembly 100 as attribute 3 of class 0x04, alternating the two messages.



Cyclic explicit messaging updates every 50 to 200 ms, without guarantee: enough for simple point-to-point moves with generous timeouts, not for real axis control.

8.6. Commissioning and Diagnostics

1. Download and check that the module reaches *Running*.
2. Online, `iSw` must be non-zero, `iAlarm` 0 and `iDcVolt` the expected voltage; the drive decodes as *Switch On Disabled*.
3. Set `cmdEnable` and follow steps 10 → 20 → 30 → 40: if the sequence stalls, the StatusWord shows where.
4. Run a homing, then a short move at low speed, with the mechanics free.
5. Only then reconnect the mechanics and raise speeds.

Symptom	What to check
Connection never *Running*	Assembly instances 100 / 101 / 102 and sizes 50 / 20 / 4
Extended status 0x0126, 0x0127, 0x0128	Wrong size of configuration, O>>T, T>>O assembly
Connection already owned	The drive accepts one Exclusive Owner: close other sessions
StatusWord always 0	P176 on the drive is not 0
Nonsensical values	COP offset off by one word, or word order reversed
Sequence stalls at *Switched On*	Hardware permit missing or latched alarm: read `iAlarm`
No motion in Profile Position	Bit 4 handshake incomplete, mode display not 1, or target outside the limits



Enabling over EtherNet/IP is not a safety function: emergency stop, STO and limit switches stay hard wired. Define also the behaviour of the drive on loss of the connection, which is configured on the drive.

Appendix A: Updating the Rockwell-Like AOI from Revision 1.0

Revision 1.1 of the Rockwell-like command blocks corrects defects that could stop a sequence or move the axis the wrong way. Importing the files is not enough: two changes alter the interface and two alter the behaviour of a running machine.

Change	Effect after the import	What to do
MM_MAM and MM_MAJ: *Decerelation* renamed *Deceleration*, both ramps from INT to DINT	The pin is left empty on every instance	Wire it again
MM_MAM: *Direction* renamed *MoveType*	The pin is left empty on every instance	Wire it again
MM_MAM: polarity of *MoveType* corrected	**Nothing visible**. An absolute move becomes incremental and vice versa	Invert the value on every instance, unless it already followed this guide
MM_MAJ: the jog no longer stops when the rung drops, as on a Logix MAJ	**Nothing visible**. The motor keeps turning at the last speed	Stop the jog with MM_MAS where the application relied on dropping the rung
Timeout, *ErrorCode* on every command block; *MaxTorque* on MM_MSO, MM_MAM, MM_MAJ; *ZeroSpeedWindow* on MM_MAS	They have defaults: old instances keep working	Wire *ErrorCode* to the HMI; raise *Timeout* for long homings
MM_MSL, new block	Not required	Use it if the application wrote the position limits into `DBS55_In` by hand

MoveType	Revision 1.0 did	Revision 1.1 does
0	Relative move	Absolute move
1	Absolute move	Relative move

Suggested order:

1. Import the files into a copy of the project. Each file holds one block, so an import does not touch the others.
2. Fix the empty *Deceleration* and *MoveType* pins, and invert *MoveType* where needed.
3. Set *Timeout* on the homing.
4. With the mechanics free, run the whole cycle: enable, homing, positioning, jog, stop, jog again. The jog, stop, jog sequence did not work in 1.0 and is the quickest proof that the update is in.
5. Only then reconnect the mechanics.



With homing method 37 the drive can complete faster than one PLC scan. Check on the bench that MM_MAH ends with *Done* and not with *ErrorCode* = 1.