ETHERCAT SET UP GUIDE

Riccardo Piccinini
Mini Motor S.P.A.

FBS-ECT-EN

Firmware Revision: 5.004

Version 5, 09-2026

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

2. EtherCAT . . . **10**
- 2.1. Overview . . . 10
- 2.2. DS301 Communication Layer . . . 10
- 2.3. Object Dictionary . . . 12

3. Application Example - Omron . . . **19**
- 3.1. Tools . . . 19
- 3.2. Overview . . . 19
- 3.3. Install DBS ESI File . . . 19
- 3.4. Profile Example . . . 21
- 3.5. CSP Example . . . 23

4. Application Example - Beckhoff . . . **27**
- 4.1. Tools . . . 27
- 4.2. Overview . . . 27
- 4.3. Profile Example . . . 29
- 4.4. CSP Example . . . 30
- 4.5. EoE Activation . . . 31

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

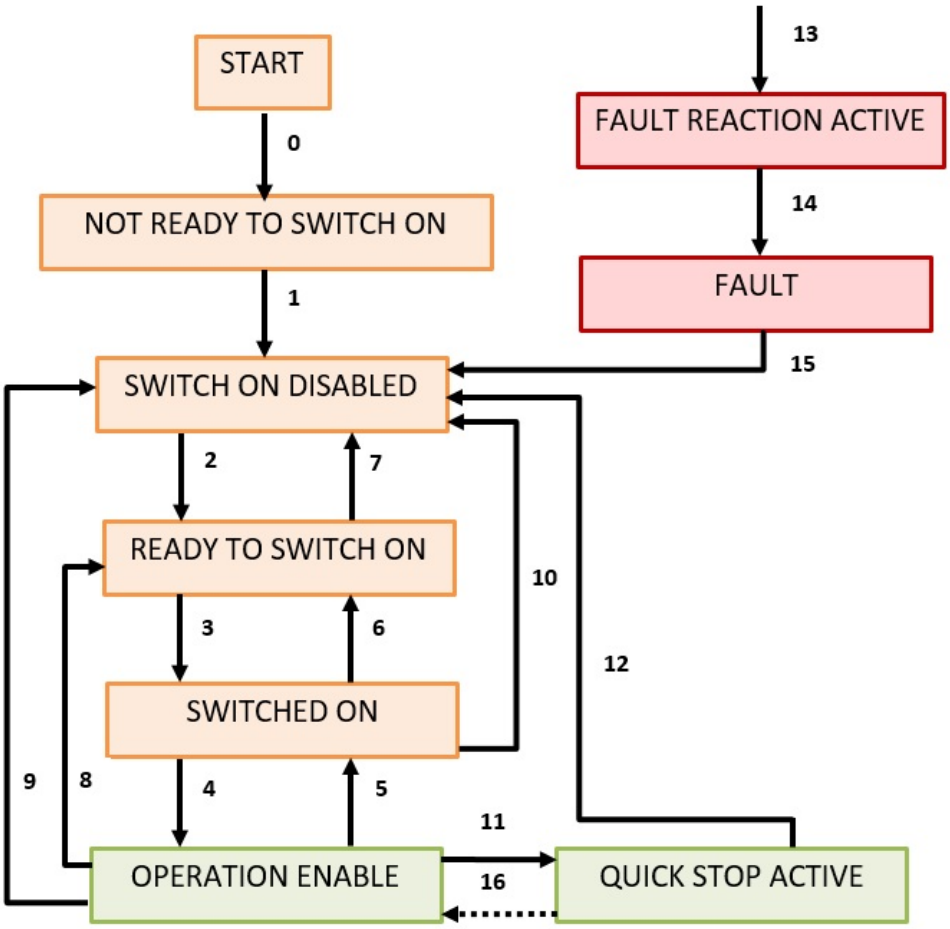
Whatever the mode, the interaction between the master (the controller) and the slave (the servo drive) always follows the same pattern:

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.

2. EtherCAT

2.1. Overview

Note: available only for drivers equipped with CANopen/Modbus RTU, CANopen over EtherCAT or Ethernet POWERLINK. (---/---/C; ---/---/ETH; ---/---/EPL).

The present document describes the adherence of the Minimotor stack to the following specification:

CANopen

- CiA DS301 v4.02 (CANopen DLL)
- CiA DS402 v3.0.1.15 (servodrive profile)

CANopen over EtherCAT:

- ETG1000-2 v1.0.1 (physical layer)
- ETG1000-3 v1.0.1 (DLL services)
- ETG1000-4 v1.0.1 (DLL protocols)
- ETG1000-5 v1.0.1 (AL services)
- ETG1000-6 v1.0.1 (AL protocols)
- CiA DS402 v3.0.1.15 (servodrive)
- ETG6010 v1.1.0 (CiA 402 impl. directives)

Ethernet POWERLINK (EPL)

- EPSG DS 301 V1.2.0 (POWERLINK DLL)

The purpose of the present document is to describe what optional features are implemented by the servo-motor CANopen, CoE or EPL.

Applicable ---/---/--- software release: ≥v4.036

2.2. DS301 Communication Layer

This paragraph describes the details of the DS301 layer of the CANopen interface. It applies only to the CANopen over can-bus implementation.

2.2.1. Physical Layer

Supported baudrate:

- 1Mbps tq=71ns SamplingPoint=11tq
- 500Kbps tq=142ns SamplingPoint=12tq

- 250Kbps tq=284ns SamplingPoint=12tq
- 125Kbps tq=571ns SamplingPoint=12tq
- 50Kbps tq=1.43us SamplingPoint=12tq
- 20Kbps tq=3.57us SamplingPoint=12tq

2.2.2. SDO protocol

- Maximum supported object size: 32bit
- Expedited transfer supported SUPPORTED
- Segmented transfer: NOT SUPPORTED
- Block transfer: NOT SUPPORTED

2.2.3. TIME protocol

The time protocol is NOT supported.

2.2.4. PDO protocol

The implementation supports up to 4 Pdo TX and 4 Pdo RX; each object can map up to 8 objects; mapping can be done only using the whole size of the object (i.e. is not possible to map 8 bit of a 32 bit object).

The following transmit type are implemented for PDO TX:

Transmission Type	Description
0	PDO is emitted when changes occur after SYNC is received
1..240	PDO emitted after SYNC, depending on SYNC message occurrence (e.g. 1 = every SYNC, 2 = every 2 SYNC, etc.)
241..251	Not supported
252	PDO emitted after RTR reception * Data sampled on SYNC * Emitted after RTR
253	PDO emitted after RTR reception * Data sampled and emitted after RTR
254	Not supported
255	PDO emitted asynchronously Two features supported for emission control: * Inhibit time (emission only when PDO content changes) * Event timer (periodical asynchronous emission)

Defaults:

- inhibit time: 100ms
- event time: 0 (periodic emission on TT=255 disabled by default)

See application profile for default PDO configuration. ==== EMCY protocol

Emergency protocol is implemented for reporting alarms from the servodrive; to control the emission of

EMCY messages, the inhibit time (object 1015h) can be modified. See the application profile for detailed list of alarms. (See paragraph 8.3)

2.2.5. NMT protocol

The NMT implementation includes standard commands for NMT state machine management. For node alive checking, the NODE GUARDING protocol is implemented; the behaviour of node guarding is set by objects 100Ch (guard time) and 100Dh (life factor).

2.2.6. HEARTBEATING

Heartbeating Protocol is supported. Register 1016h defines the time in multiples of 1ms for the Heartbeat protocol consumer. Register 1017h determines the time in multiples of 1ms for the producer of the protocol. Object 1029h determines the behaviour when the error is triggered.

2.3. Object Dictionary

Object dictionary is divided in the following sections:

Range Objects	Functions
1000h .. 1FFFh	Communication profile area
2000h .. 21FFh	Manufacturer parameter area
3000h .. 3FFFh	Manufacturer specific object area
6000h .. 67FFh	Profiled objects

Other areas are unused; the object list is common between CANopen, CoE and EPL implementation.

2.3.1. Communication area

The following table shows the list of communication area objects; since some objects are relevant only for CANopen or CoE implementation, the table shows in which fieldbus the object is relevant.

Index	Description	CANopen	CoE	EPL
1000	Device type	X	X	X
1001	Error register	X	X	X
1003	Predefined error	X	X	
1006	Cycle length			X
100C	Guard time	X		
100D	Life factor	X		
1010	Store parameters, see Saving parameters to non-volatile memory	X	X	X
1014	EMCY cob-id	X		

Index	Description	CANopen	CoE	EPL
1015	EMCY inhibit time	X	X	
1016	Consumer Heartbeat Time	X		
1017	Producer Heartbeat Time	X		
1018	Identity object	X	X	X
1020	CFM Verify Configuration			X
1029	Error Behaviour	X		
1030	NMT Interface Group 0h			X
1300	SDO Sequ Layer Timeout			X
1400	PDO RX 1 – config [1]	X	X	X
1401	PDO RX 2 – config	X		
1402	PDO RX 3 – config	X		
1403	PDO RX 4 – config	X		
1600	PDO RX 1 – mapping [1]	X	X	X
1601	PDO RX 2 – mapping	X		
1602	PDO RX 3 – mapping	X		
1603	PDO RX 4 – mapping	X		
1800	PDO TX 1 – config [1]	X	X	X
1801	PDO TX 2 – config	X		
1802	PDO TX 3 – config	X		
1803	PDO TX 4 – config	X		
1A00	PDO TX 1 – mapping [1]	X	X	X
1A01	PDO TX 2 – mapping	X		
1A02	PDO TX 3 – mapping	X		
1A03	PDO TX 4 – mapping	X		
1C00	Sync manager types		X	
1C0A	DLL_CnCollision REC			X
1C0B	DLL_CnLoss REC			X
1C0C	DLL_CnLossSoC_REC			X
1C0D	DLL_CnLossSoA REC			X
1C0E	DLL_CnSpCJitter REC			X
1C0F	DLL_CnCRCError REC			X
1C10	Sync manager 0 config		X	
1C11	Sync manager 1 config		X	

Index	Description	CANopen	CoE	EPL
1C12	Sync manager 2 config		X	
1C13	Sync manager 3 config [1]		X	
	DLL_CnSocJitterRange [1]			X
1C14	DLL_CnLossOfSocTolerance			X
1C32	Output sync parameters		X	
1C33	Input sync parameters		X	
1E40	NWL_IpAddrTable_0h_REC			X
1E4A	NWL_IpGroup_REC			X
1F82	NMT_FeatureFlags_U32			X
1F83	NMT_EPLVersion_U8			X
1F8C	NMT_CurrNMTState_U8			X
1F93	NMT_EPLNodeID_REC			X
1F98	NMT_CycleTiming_REC			X
1F99	NMT_CNBasicEthernetTimeout_U32			X
1F9A	NMT_HostName_VSTR			X
1F9E	NMT_ResetCmd_U8			X

Saving parameters to non-volatile memory

Writing an object in the 2000h … 21FFh range 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 *Store parameters* object `1010h.1`.

2.3.2. Manufacturer parameter area

The object range 2000h … 21FFh allows to access servodrive parameters; the following table summarize the details of a parameter object:

Object details

Field	Value
Subidx	0
Object type	16bit, signed integer
Access	Read/Write
PDO mappable	NO

Object dictionary for manufacturer parameters

Index	Description
2000h	Parameter 0
2001h	Parameter 1
…	…
21FFh	Parameter 511

For details of the parameter meaning and encoding refer to the servodrive manual. IMPORTANT: Writing objects in the 2000h … 21FFh range will alter only the RAM image of the parameter; to actually save a parameter to non-volatile storage a 32 bit write of the value `65766173h` ("SAVE") to object `1010h.1` needs to be performed, as described in Saving parameters to non-volatile memory.

2.3.3. Manufacturer Specific Area

This area contains several object that allows to read/write manufacturer specific data.

Index	Access	Data type	PDO mappable	Description
3000h	RO	INT16	YES	Ain0 value (-32768 to 32768)
3001h	RO	UINT16	YES	Digital input bits 0–4
3002h	RW	UINT16	YES	Digital output bit0
3003h	RO	INT16	YES	Heatsink temperature (x0.1 °C)
3005h	RO	INT16	YES	Motor temperature (x0.1 °C)
3008h	RO	UINT16	YES	Actual alarm code (not EMCY)
3009h	RO	UINT	NO	Events (0–16, see section 6.1)
3010h	RO	UINT16	YES	Power stage tension (x0.1 V)
3011h	RO	INT16	YES	Motor IQ current (mA)
3012h	RW	UINT16	YES	Motor IQ limit (mA)
3014h	RO	INT16	YES	Output power (1W/lsb)
3020h	RW	INT32	YES	Touchprobe positioning offset
3021h	RW	UINT32	YES	Touchprobe positioning modulo
3030h	RO	UINT16	YES	Acceleration RMS modulus (milli-G)
3031h	RO	UINT16	YES	Acceleration RMS x (milli-G)
3032h	RO	UINT16	YES	Acceleration RMS y (milli-G)
3033h	RO	UINT16	YES	Acceleration RMS z (milli-G)
3034h	RO	UINT16	YES	Acceleration DC x (milli-G)
3035h	RO	UINT16	YES	Acceleration DC y (milli-G)
3036h	RO	UINT16	YES	Acceleration DC z (milli-G)

Index	Access	Data type	PDO mappable	Description
3037h	RO	INT16	YES	Instantaneous Acceleration x (milli-G)
3038h	RO	INT16	YES	Instantaneous Acceleration y (milli-G)
3039h	RO	INT16	YES	Instantaneous Acceleration z (milli-G)
3070h	RW	INT64	YES	Set point time, ns of the CIP Sync (PTP) clock. Used only by EtherNet/IP with CIP Sync
3071h	RO	INT64	YES	Position time, ns of the CIP Sync (PTP) clock. Used only by EtherNet/IP with CIP Sync
3072h	RO	UINT16	YES	Global time (CIP Sync) status: bit0 PTP synchronised, bit1 time valid, bit2 sync0 ok, bit3 locked to the time base, bit4 set point time received. Used only by EtherNet/IP with CIP Sync

2.3.4. Profiled Objects

The range 6000h … 67FFh of objects are defined as specified by CiA402; refer to the application area for further information.

Index	Access	Type	PDO mappable	Description
6040h	RW	U16	YES	Control word
6041h	RO	U16	YES	Status word
605Ah	RW	U16		Quickstop option code
605Bh	RW	U16		Shutdown option code
605Ch	RW	U16		Disable operation option code
605Dh	RW	U16		Halt option code
605Eh	RW	U16		Fault reaction option code
6060h	RW	S8	YES	Mode of operation
6061h	RO	S8	YES	Mode of operation display
6062h	RO	S32	YES	Position demand value
6064h	RO	S32	YES	Position actual value
6065h	RW	U32		Following error window
6066h	RW	U16		Following error time
6067h	RW	U32		Position window
6068h	RW	U16		Position window time
606Bh	RO	S32	YES	Velocity demand
606Ch	RO	S32	YES	Velocity actual value
606Dh	RW	U16		Velocity window

Index	Access	Type	PDO mappable	Description
606Eh	RW	U16		Velocity window time
606Fh	RW	U16		Velocity threshold
6070h	RW	U16		Velocity threshold time
6071h	RW	S16	YES	Target torque
6072h	RW	U16	YES	Max torque
6073h	RW	U16	YES	Max current
6074h	RO	S16	YES	Torque demand value
6075h	RO	U32	YES	Motor rated current
6076h	RO	U32	YES	Motor rated torque
6077h	RO	S16	YES	Torque actual value
6078h	RO	S16	YES	Current actual value
6079h	RO	U32	YES	DC link voltage
607Ah	RW	S32	YES	Target position
607Bh.0	RO	U8	YES	Position Range Limit – number of subindex
607Bh.1	RW	S32	YES	Negative limit
607Bh.2	RW	S32	YES	Positive limit
607Ch	RW	S32	YES	Home offset
607Dh.0	RO	U8	YES	Software position limit – number of subindex
607Dh.1	RW	S32	YES	Negative limit
607Dh.2	RW	S32	YES	Positive limit
607Eh	RW	U8		Polarity
607Fh	RO	U32	YES	Max speed
6080h	RO	U32	YES	Max speed (duplicate index)
6081h	RW	U32	YES	Profile velocity
6083h	RW	U32	YES	Profile acceleration
6084h	RW	U32	YES	Profile deceleration
6085h	RW	U32	YES	Quick stop deceleration
6087h	RW	U32	YES	Torque slope
6098h	RW	S8	YES	Homing method
6099h.0	RO	U8	YES	Homing speed – number of subindex
6099h.1	RW	U32	YES	Switch speed
6099h.2	RW	U32	YES	Index speed

Index	Access	Type	PDO mappable	Description
609Ah	RW	U32	YES	Homing acceleration
60B0h	RW	S32	YES	Position offset
60B1h	RW	S32	YES	Velocity offset
60B2h	RW	S16	YES	Torque offset
60B8h	RW	U16	YES	Touch-probe function
60B9h	RW	U16	YES	Touch-probe status
60BAh	RW	S32	YES	Touch-probe positive edge latched position
60BBh	RW	S32	YES	Touch-probe negative edge latched position
60C0h	RO	S16	NO	IpData submode selection
60C1h.0	RO	U8	YES	IpDataRecord – number of subindex
60C1h.1	RW	S32	YES	IpData record
60C2h.0	RO	U8		Interpolation time period – number of subindex
60C2h.1	RW	S8		Ip time units
60C2h.2	RW	S8		Ip time index
60F2h	RW	U16	YES	Position Option Code
60F4h	RO	U32	YES	Following Error
60FDh	RO	U32	YES	Digital inputs
60FEh.0	RO	U8	YES	Digital outputs – number of subindex
60FEh.1	RW	U32	YES	Physical outputs
60FEh.2	RW	U32	YES	Output mask
60FFh	RW	S32	YES	Target velocity
6502h	RO	U32	NO	Supported mode of operation

[1] PDO config and mapping have different encoding for CoE/CANopen and EPL. Please check DLL manual for details.

3. Application Example - Omron

3.1. Tools

Sysmac Studio Profile Example

Sysmac Studio CSP Example

DBS/FC BSI Interface Software

3.2. Overview

This section shows the creation of a Mini Motor Integrated Drive application over EtherCAT using an Omron controller, configured and programmed in the Sysmac Studio development environment.

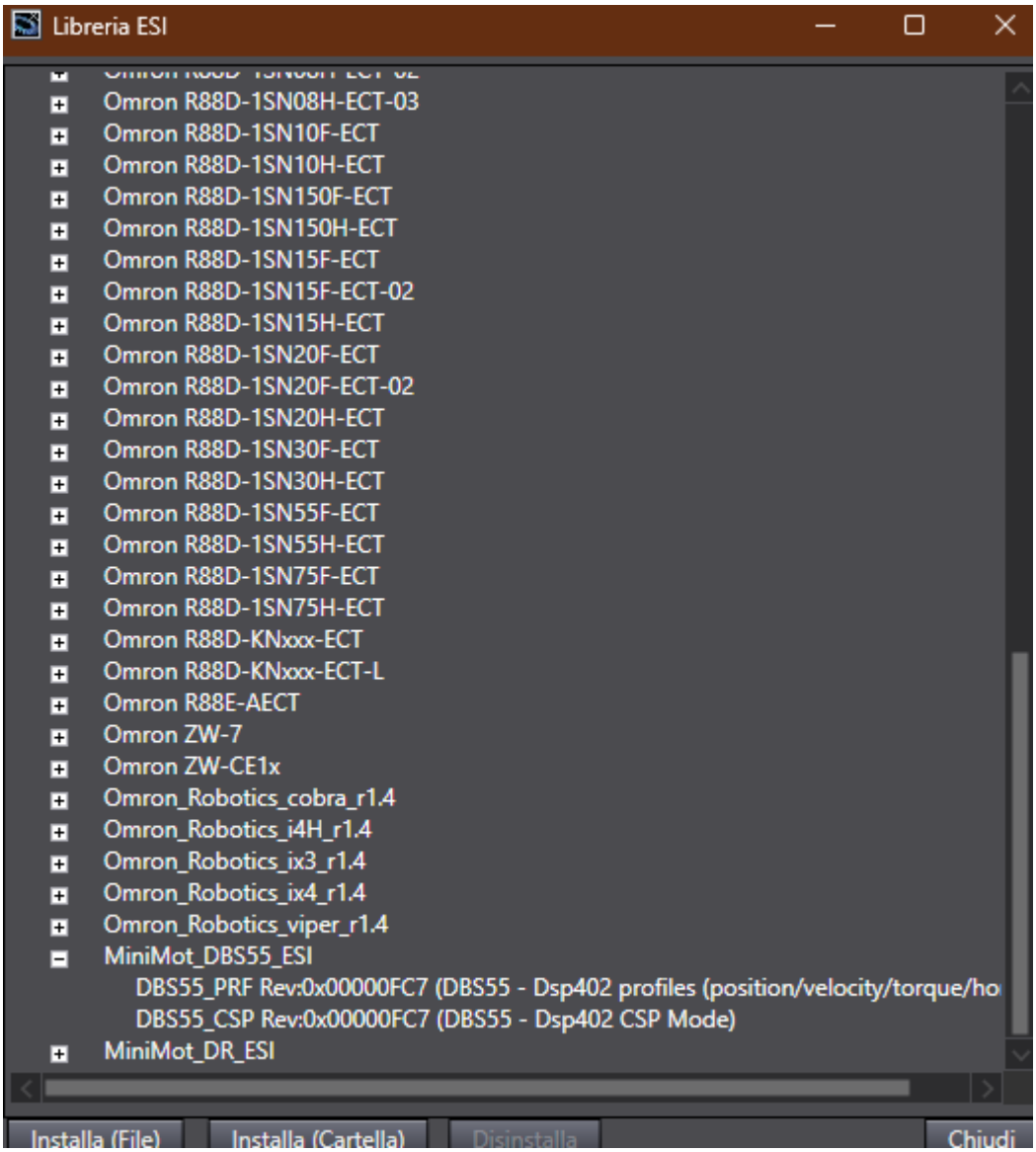
References:

- DBS/FC firmware version 5.004 or greater
- Sysmac Studio V 1.45

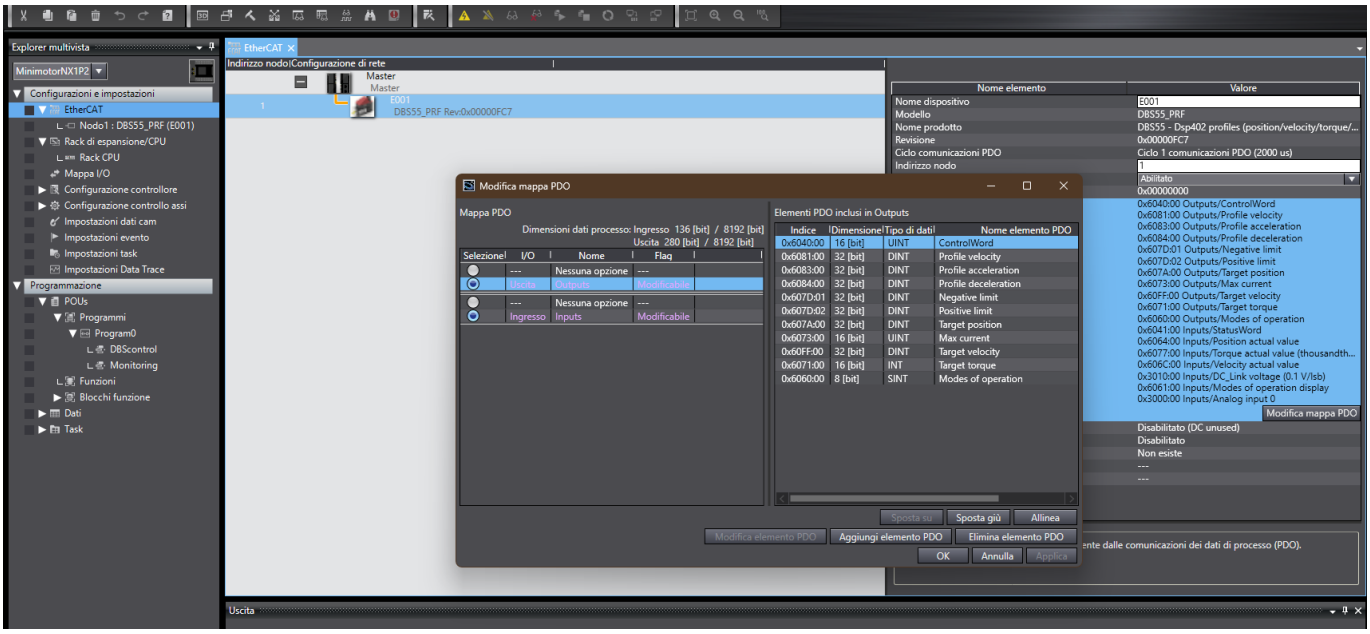
3.3. Install DBS ESI File

- Get the ESI file from the \MM-DBS_v5.xxx\DeviceDesc\EtherCAT folder of the BSI Interface Software.
- Copy it into C:\Program Files\OMRON\Sysmac Studio\IODeviceProfiles\EsiFiles\UserEsiFiles.

- Install the ESI file from the library (right click on the PLC in the EtherCAT node).



- If you want to use the motor as a node, use the DBS55_PRF ESI file.
- If you want to use the motor as an electrical axis, use the DBS55_CSP ESI file. The motor must be ordered with the CSP option for this to work and parameter P198 must be set on 1-CSP (default on CSP motors).
- In the EtherCAT field add the motor, assign the node and build the PDO. In the examples we will have pre-built PDOs to show all motor functions.



3.4. Profile Example

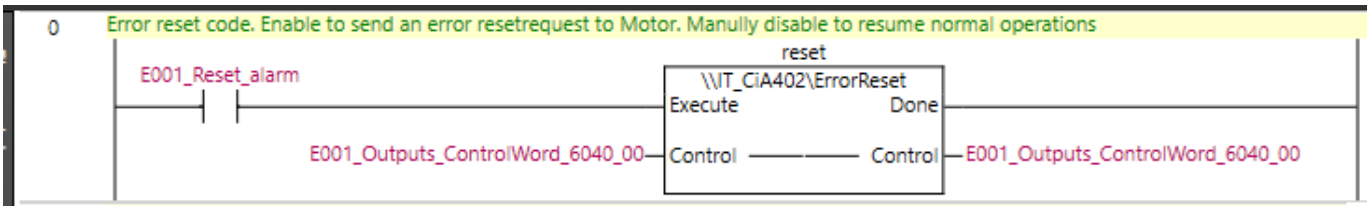
The Profile Example program shows the motor working with the CANopen DS402 Profile Velocity, Profile Position, Profile Torque and Profile Homing modes, as described by the DS402 standard. You can download the DS402 library used in the example separately.

3.4.1. DBS Control

In this part of the program you have all of the ways to control the motor following the DS402 supported profiles. You can look inside the non-library block to see which data is transferred to the motor.

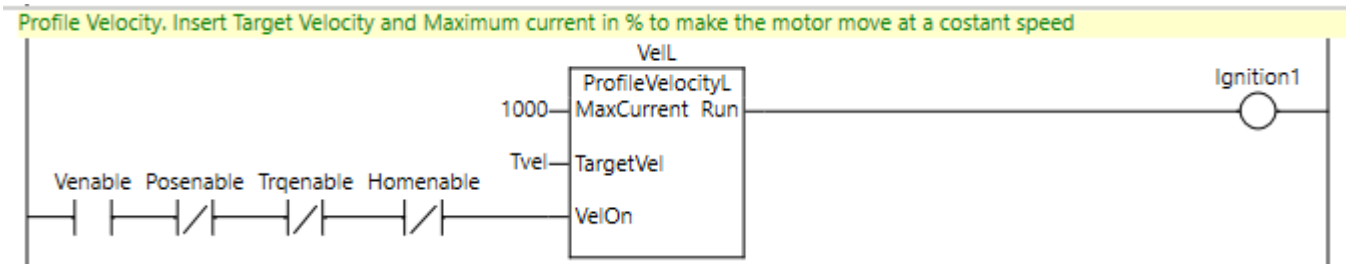
Alarm Reset

This rung is used to send an error reset to the motor. Used to clear alarms.



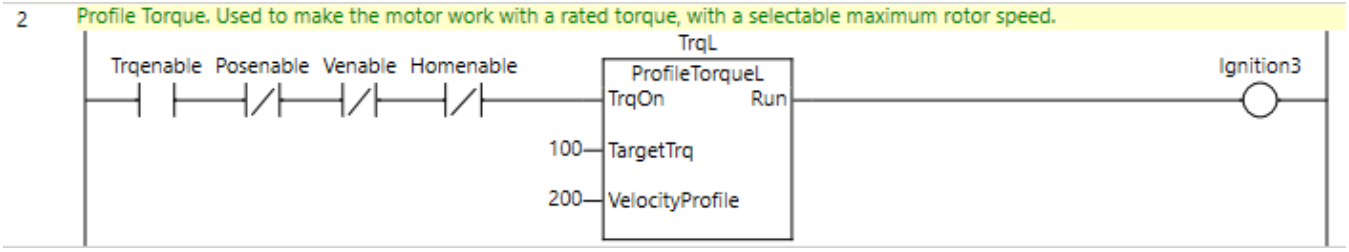
Profile Velocity

In profile velocity the motor keeps the target speed using up to the maximum current. Maximum Current is expressed as a percentage of rated current.



Profile Torque

In the profile torque section the motor will use up to the target torque, keeping a maximum profile speed. Target Torque is expressed as a percentage of rated torque.

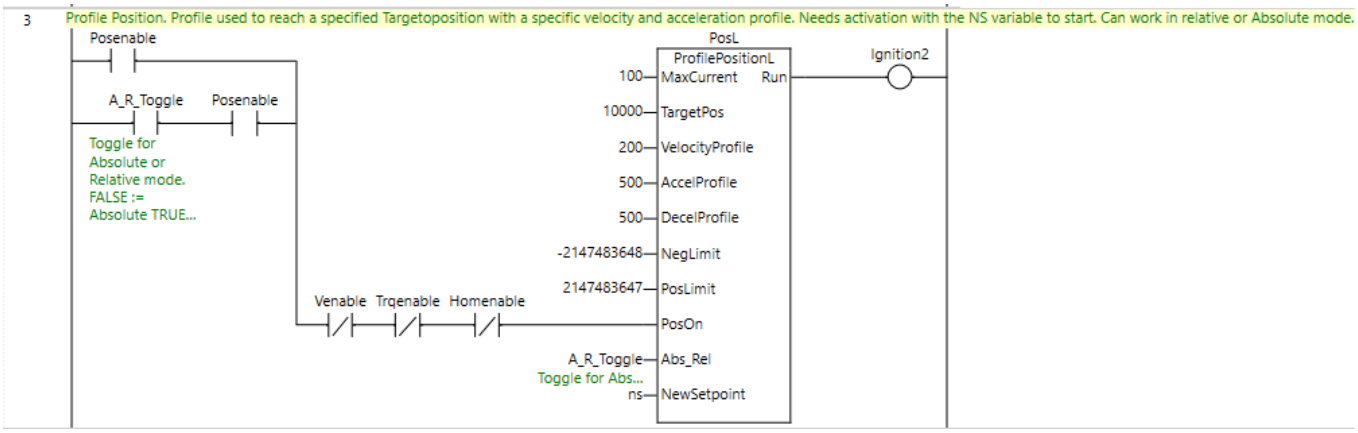


Profile Position

The profile position takes the profile velocity together with acceleration and deceleration, all in rpm and rpm/s, and uses them to generate a trapezoidal trajectory going from the actual position to the target position, using up to the maximum current. The software limits are used as software end of travel. At their maximum value (signed 32 bit register) they are disengaged.

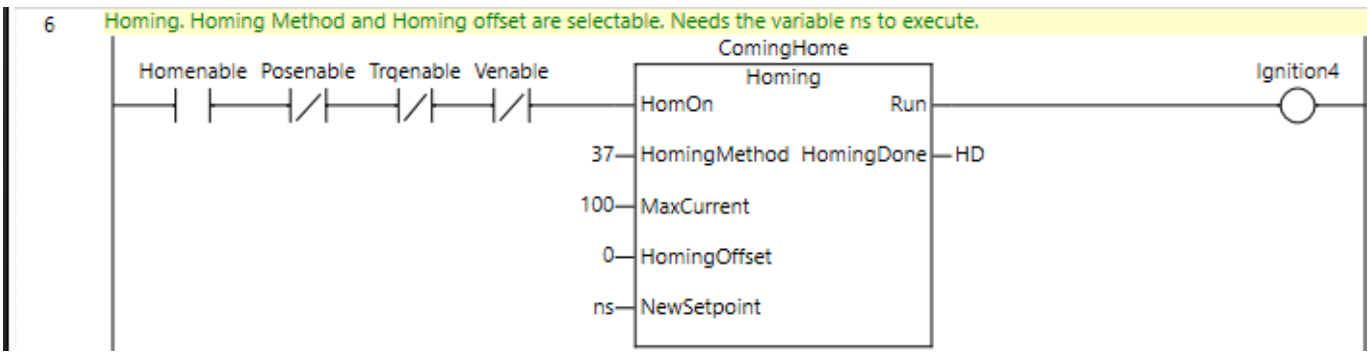
After enabling the block, the motor will enable but will not move. To do so, it requires a rising edge on the New Setpoint bit of the control word, handled by rung 4. Activating this bit will make the motor act on the target position and start moving.

The HALT command is used to momentarily stop the motor command for as long as the halt command is held. Once set to 0 again, the motor will resume the previous command, be it a velocity command or a positioning.



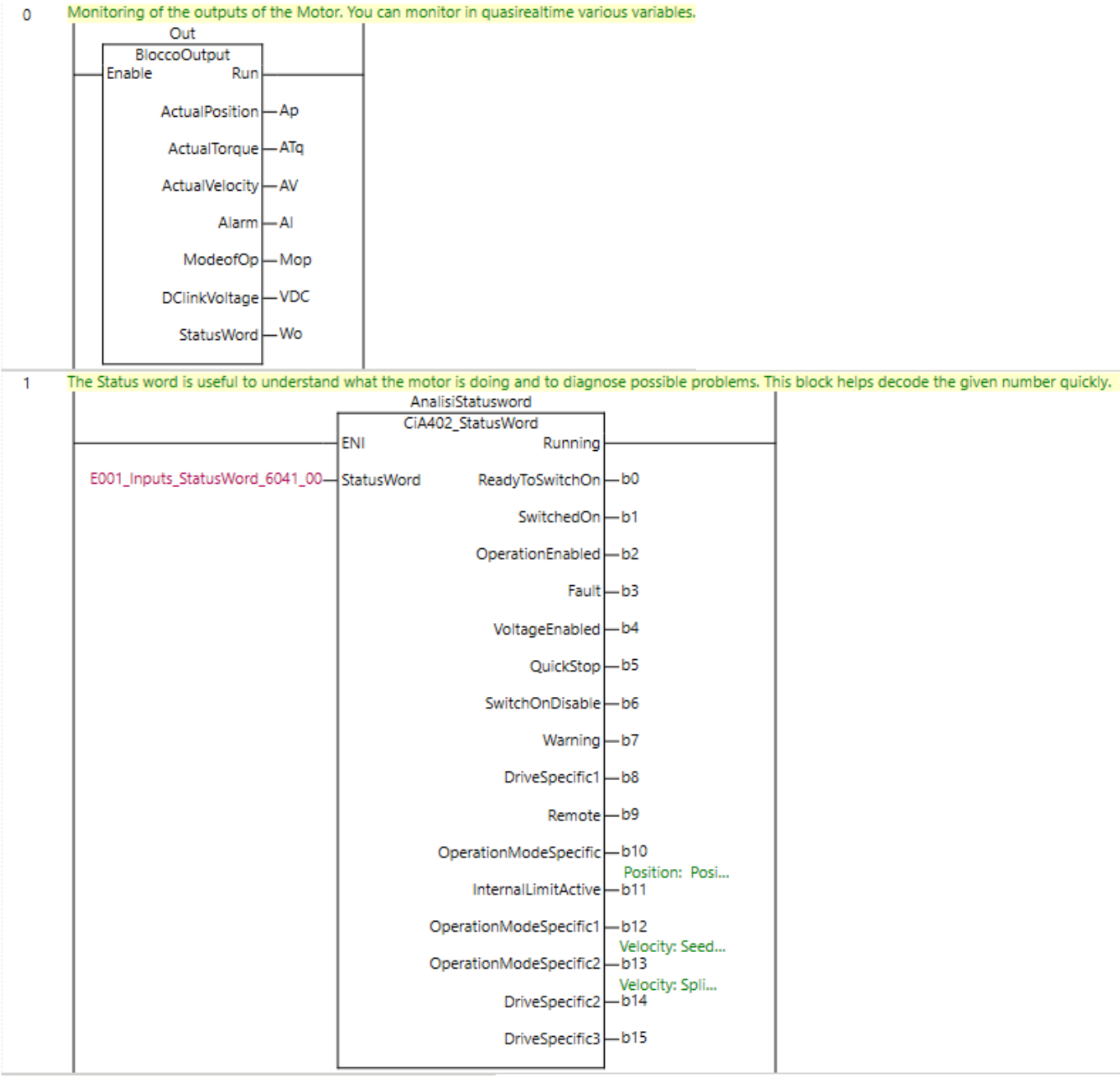
Homing

The homing procedure makes it possible to home the motor position. The default homing method here is to set the current position at 0. Check the manual for different homing procedures. To start the homing you need to activate the New Setpoint in rung 4, same as the profile position.



3.4.2. Monitoring

In the monitoring part you can check all data coming from the motor. In rung 0 you get the macro quantities, like speed, position and torque used. In rung 1 you can read each single bit of the status word to get a proper understanding of the state of the motor.

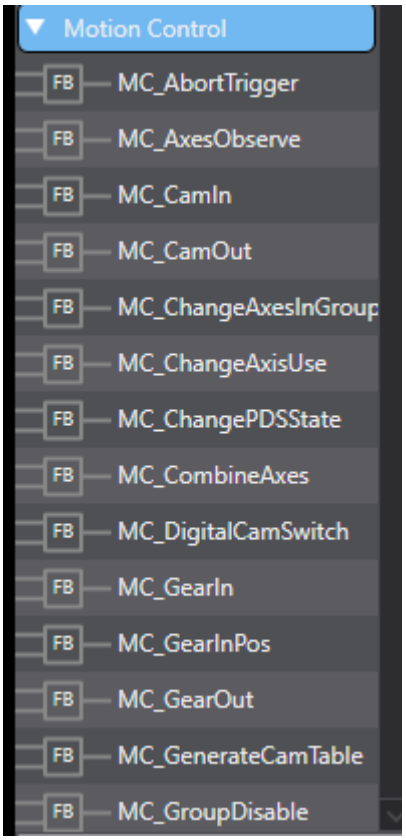


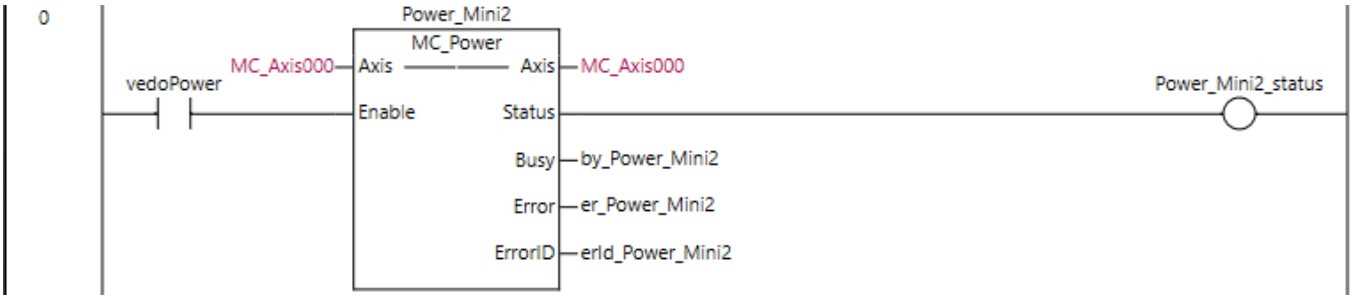
3.5. CSP Example

In the CSP example, the Motion Control blocks directly from Omron are used to get the motor running.

Power

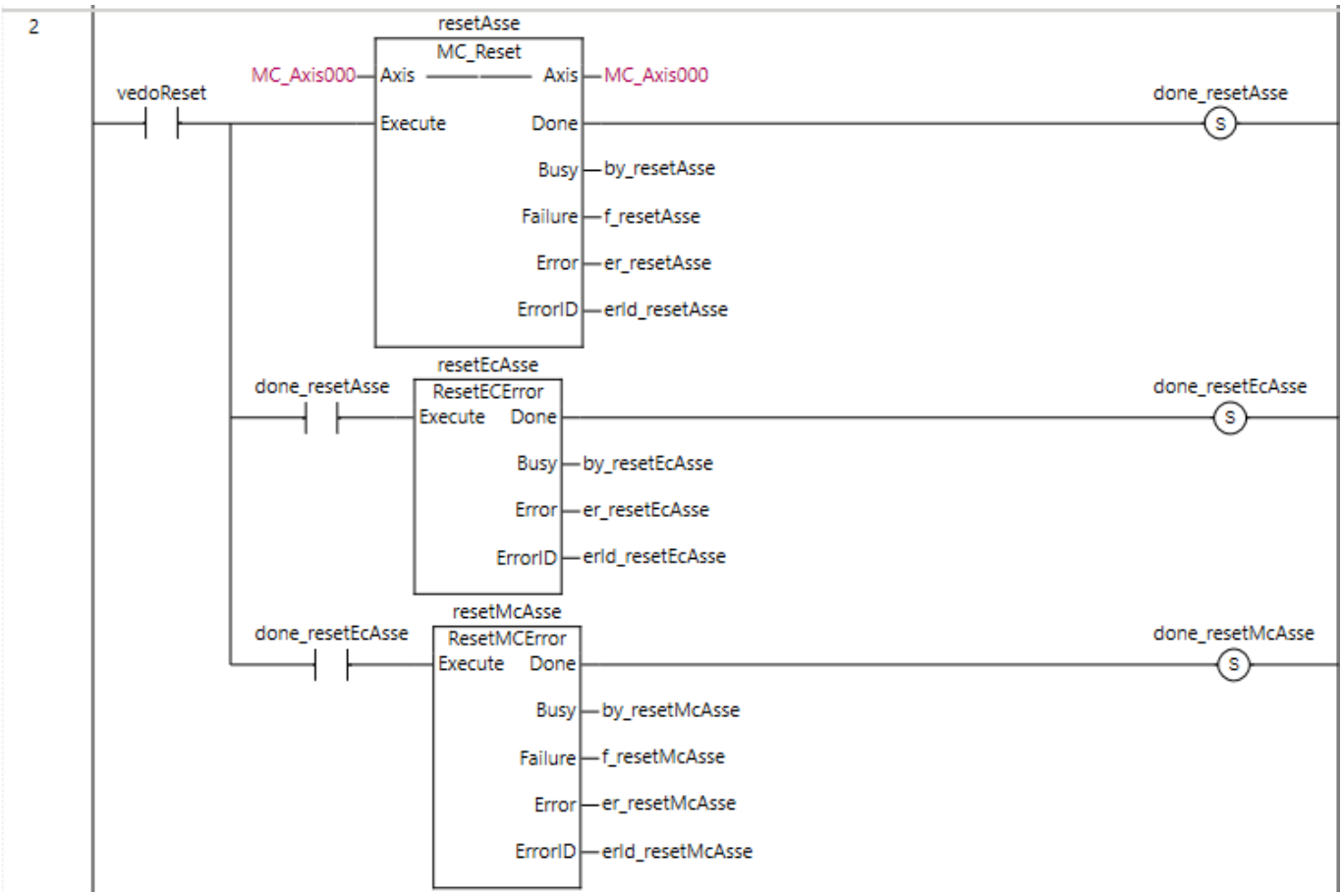
The first rung is used to enable the system. It uses the Motion Control library provided by Omron.





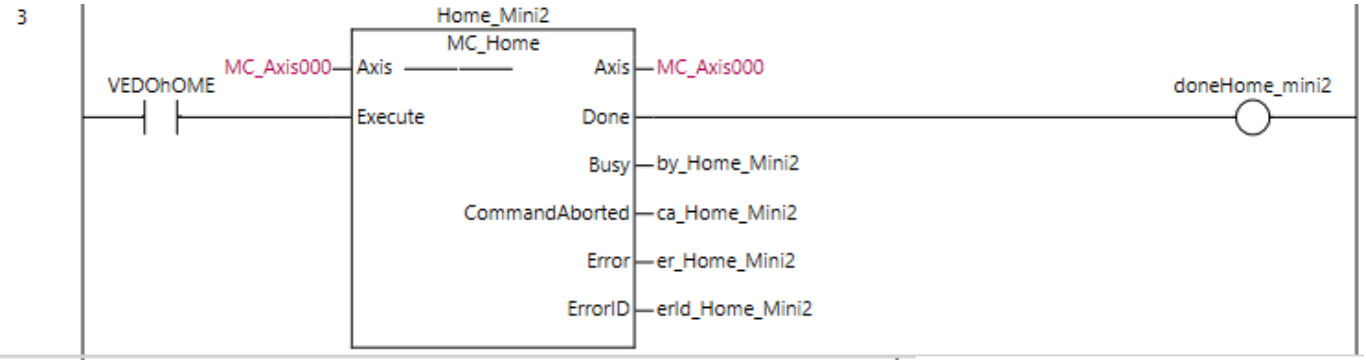
Reset

This rung is used to reset the axis.



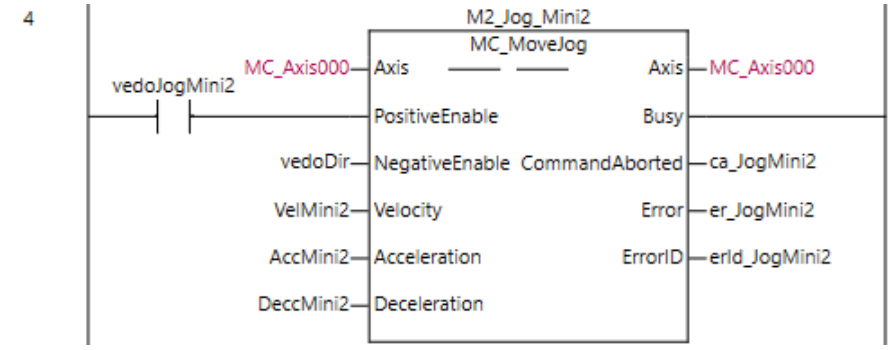
Home

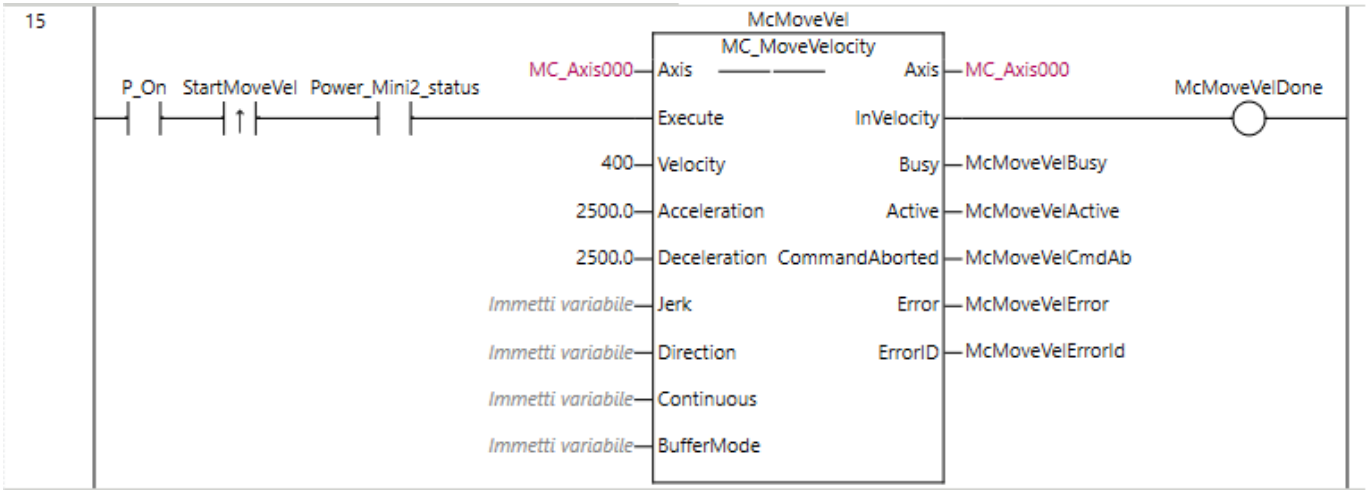
This rung is used to home the axis.

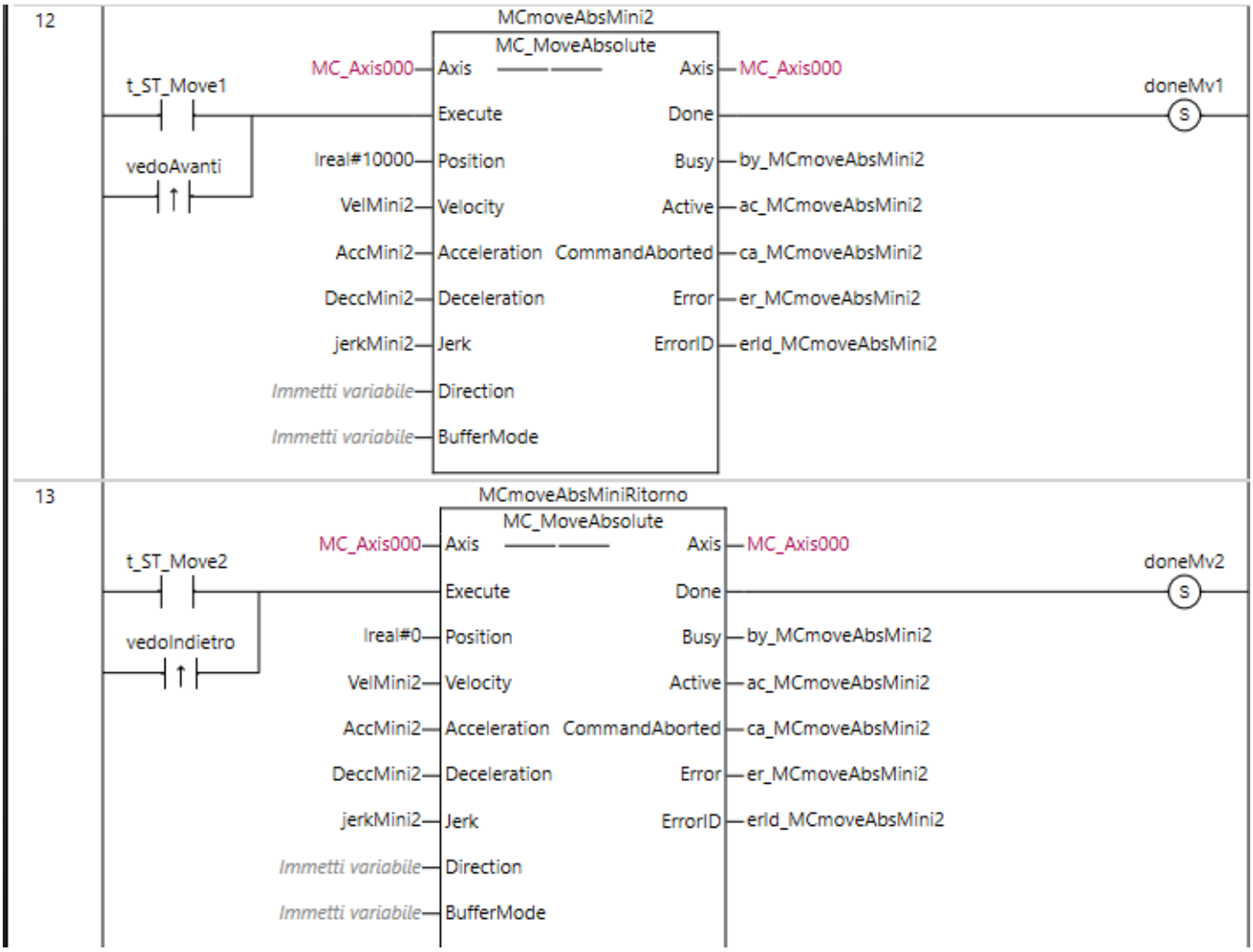


Movement

The movement in this example is shown with the Jog command, with Move Velocity and with positioning between two positions.




15
McMoveVel
MC_MoveVelocity
P_On
StartMoveVel
Power_Mini2_status
MC_Axis000
Axis
Execute
400
Velocity
2500.0
Acceleration
2500.0
Deceleration
Immetti variabile
Jerk
Immetti variabile
Direction
Immetti variabile
Continuous
Immetti variabile
BufferMode
Axis
MC_Axis000
InVelocity
Busy
McMoveVelBusy
Active
McMoveVelActive
CommandAborted
McMoveVelCmdAb
Error
McMoveVelError
ErrorID
McMoveVelErrorId
McMoveVelDone


12
MCmoveAbsMini2
MC_MoveAbsolute
t_ST_Move1
vedoAvanti
MC_Axis000
Axis
Execute
lreal#10000
Position
VelMini2
Velocity
AccMini2
Acceleration
DeccMini2
Deceleration
jerkMini2
Jerk
Immetti variabile
Direction
Immetti variabile
BufferMode
Axis
MC_Axis000
Done
doneMv1
S
Busy
by_MCmoveAbsMini2
Active
ac_MCmoveAbsMini2
CommandAborted
ca_MCmoveAbsMini2
Error
er_MCmoveAbsMini2
ErrorID
erId_MCmoveAbsMini2
13
MCmoveAbsMiniRitorno
MC_MoveAbsolute
t_ST_Move2
vedoIndietro
MC_Axis000
Axis
Execute
lreal#0
Position
VelMini2
Velocity
AccMini2
Acceleration
DeccMini2
Deceleration
jerkMini2
Jerk
Immetti variabile
Direction
Immetti variabile
BufferMode
Axis
MC_Axis000
Done
doneMv2
S
Busy
by_MCmoveAbsMini2
Active
ac_MCmoveAbsMini2
CommandAborted
ca_MCmoveAbsMini2
Error
er_MCmoveAbsMini2
ErrorID
erId_MCmoveAbsMini2

4. Application Example - Beckhoff

4.1. Tools

TwinCAT 3 Profile Example

TwinCAT 3 CSP Example

DBS/FC BSI Interface Software

4.2. Overview

This section shows the creation of a Mini Motor Integrated Drive application over EtherCAT using a Beckhoff controller, configured and programmed in TwinCAT 3 (TcXaeShell, Visual Studio shell).

References:

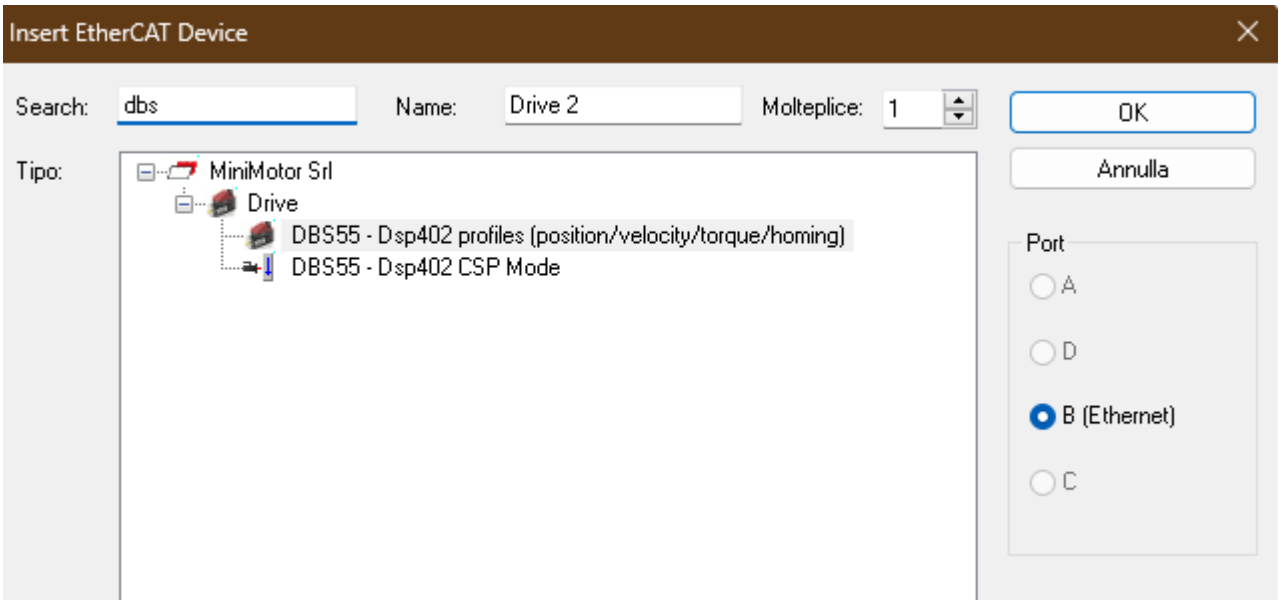
- TcXaeShell 15.0.28307.1300 D15.9

4.2.1. Install ESI File

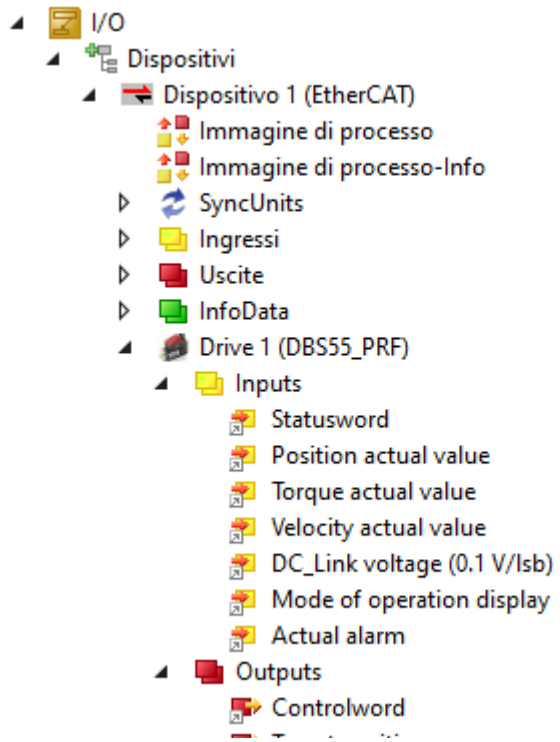
To install the ESI file, simply copy it from the Device Description folder of the BSI Interface Software to the \TwinCAT\3.1\Config\Io\EtherCAT folder.

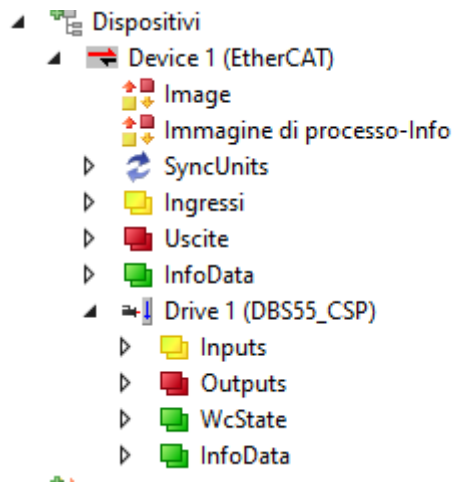
4.2.2. Import Motor and PDO Configuration

Simply add a Profile device or a CSP device under the EtherCAT device.

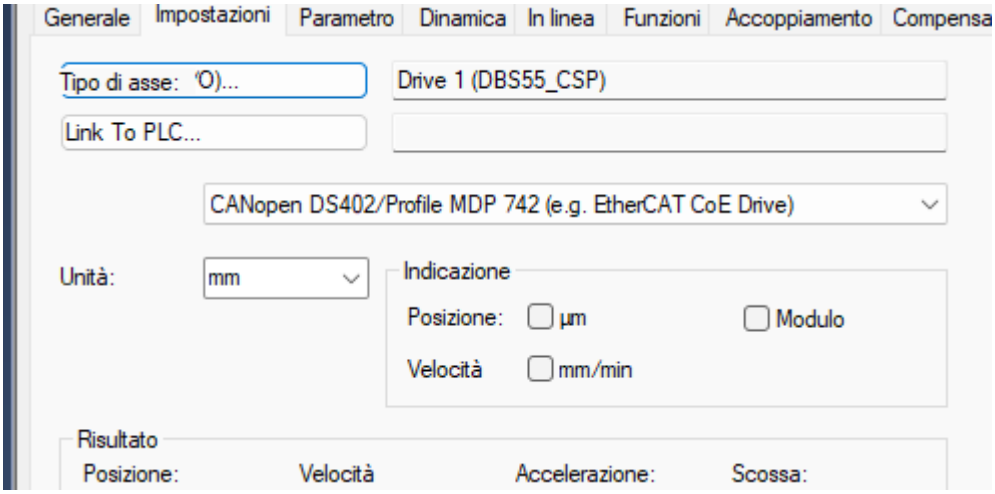


Depending on whether you need to command the motor as a node using Profile Speed / Torque / Position, or use it as an electrical axis in interpolated mode, choose the corresponding device.

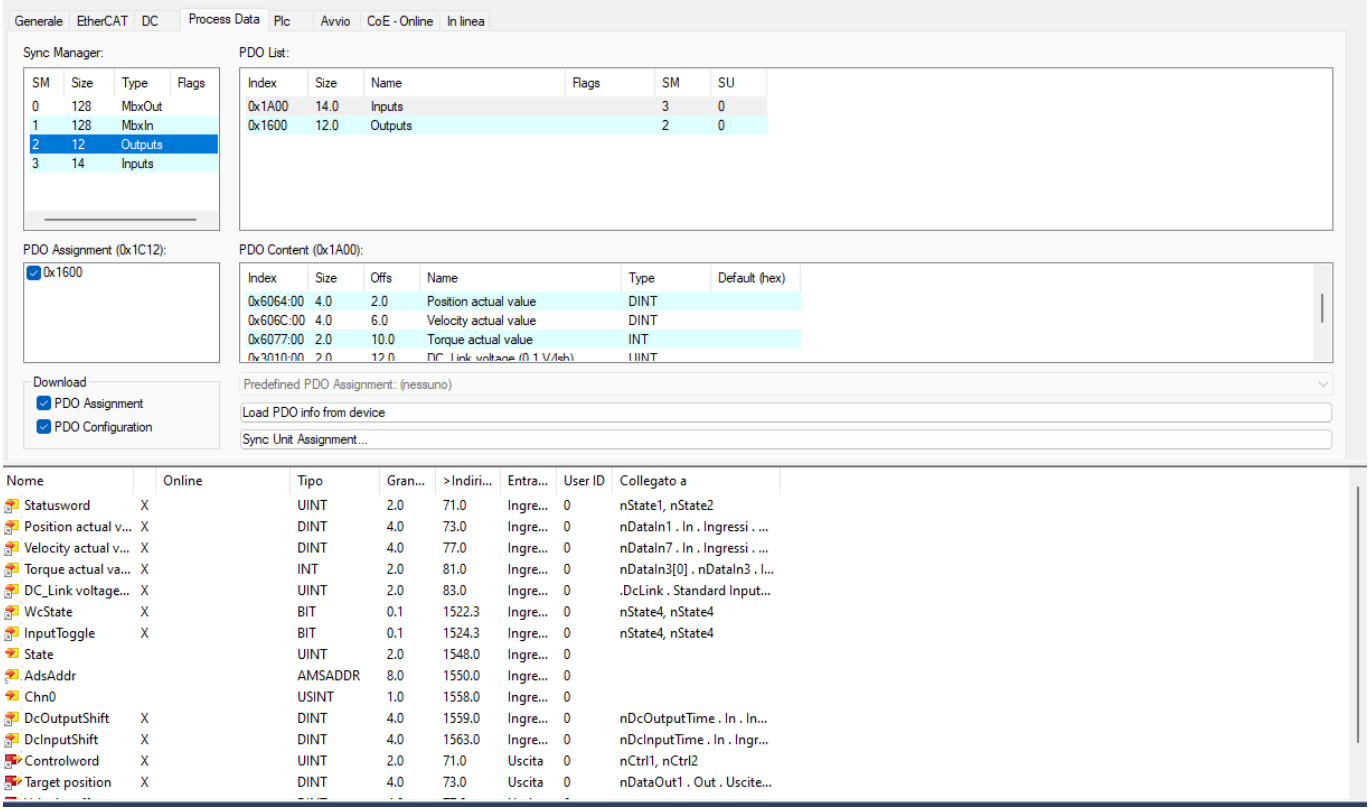




In interpolated mode it can be linked to an NC axis with the CANopen DS402 / Profile MDP 742 command profile.



In the device properties, in the Process Data tab, you can build your cyclic data exchange to suit your needs.

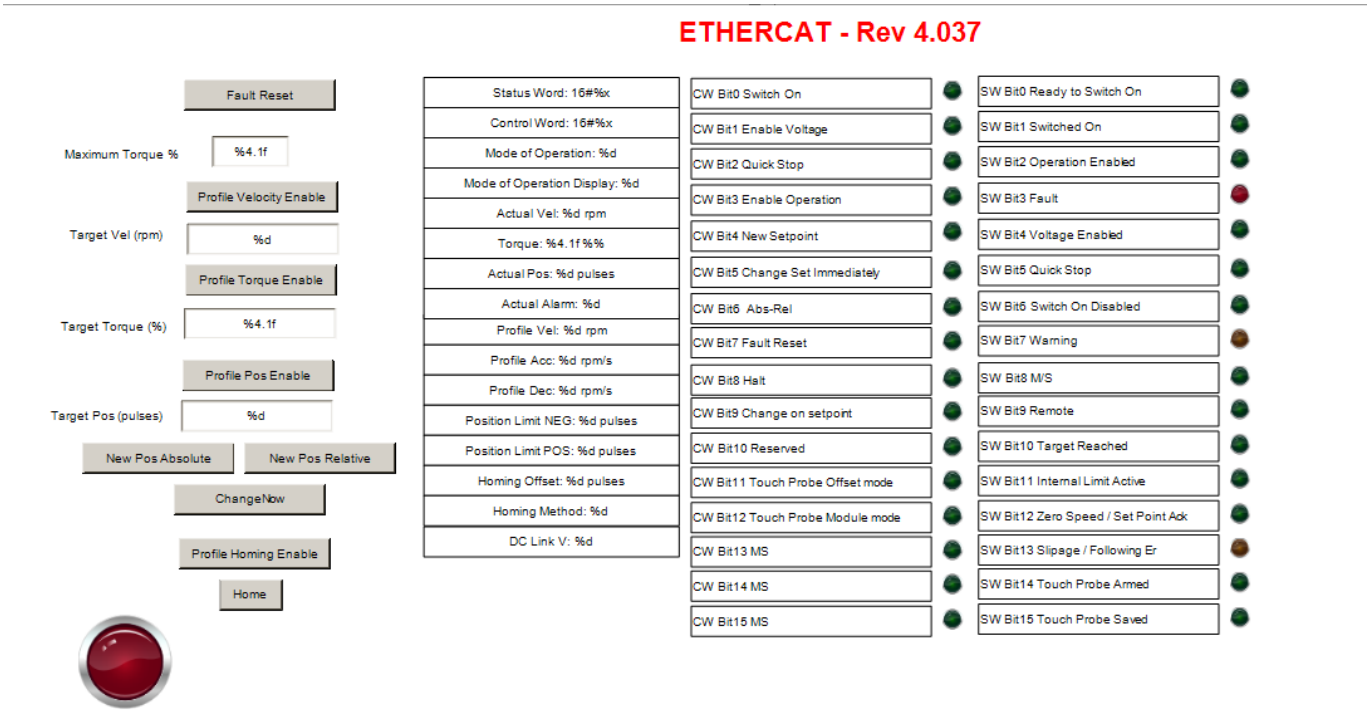


4.3. Profile Example

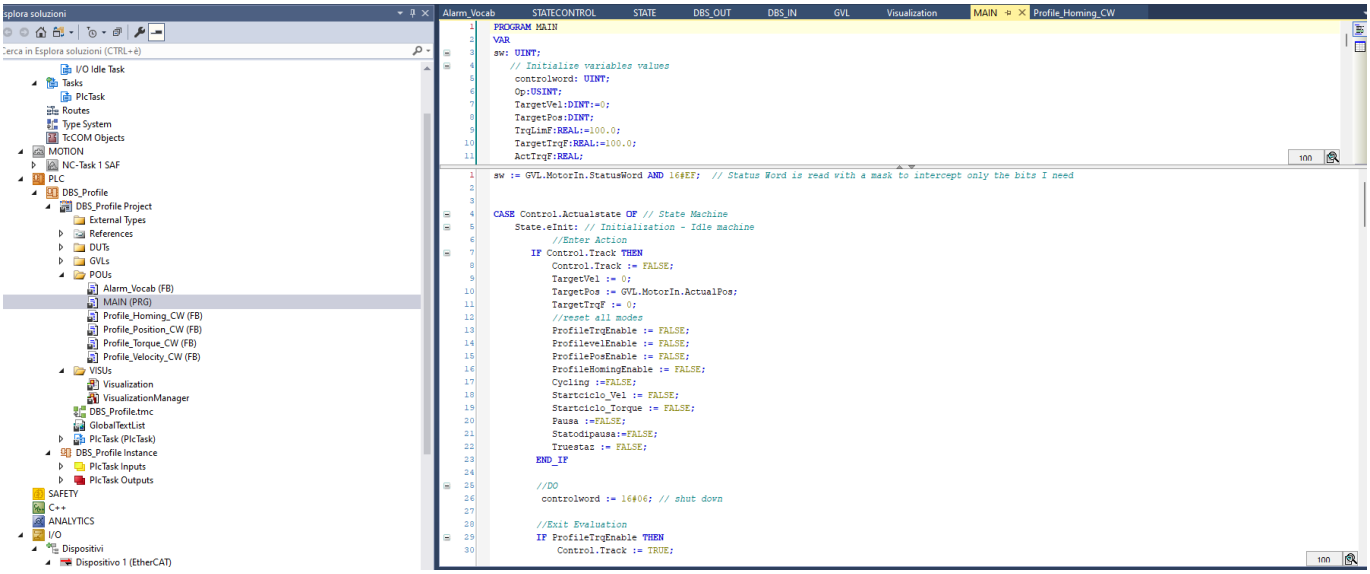
In the profile example provided, the system is already rigged to work with a single motor and to show all the possible basic functions of the drive. The PDO is already built with all the necessary variables to test all the available profiles.

In the visualization provided you can check the Control Word and Status Word state at every step of the test, and you have two main command columns. The one on the far left has all the keys for enabling each operation profile, with the relevant target value below the button. In the central columns you can read all kinds of variables and write the relevant inputs, such as the acceleration and deceleration used to build the

trajectory.

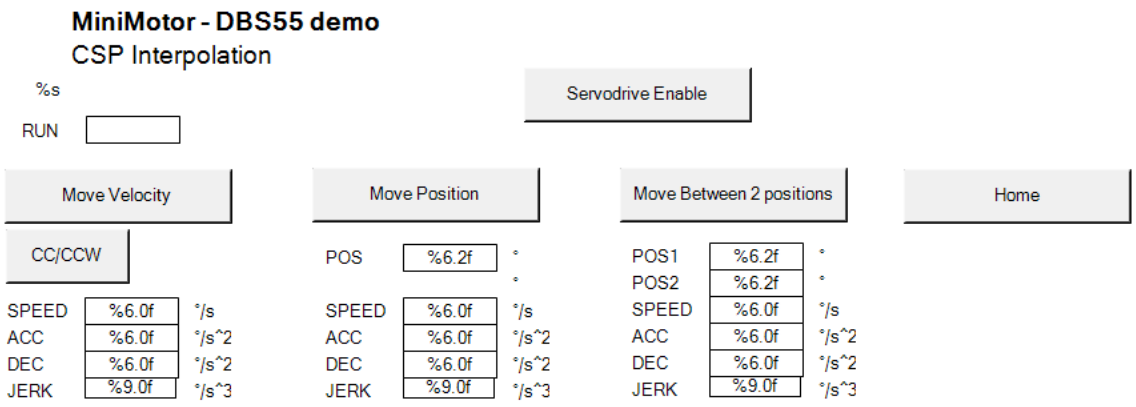


The program is handled by the state machine built in the MAIN (PRG), and the satellite Function Blocks which handle the control word / status word dialogue between the motor and the PLC to transition to the correct state for operations.



4.4. CSP Example

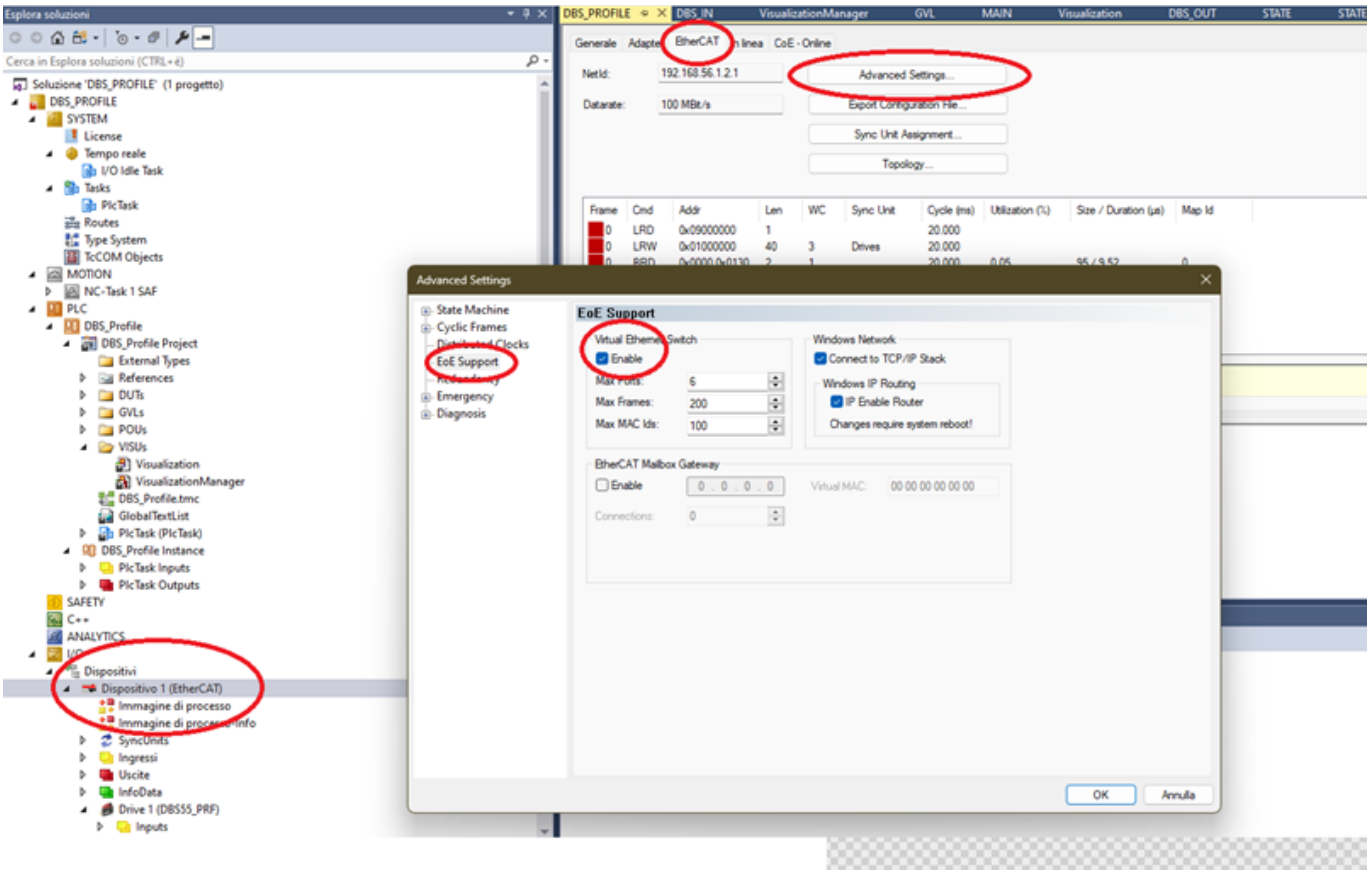
In the CSP example the motor is commanded as an electrical NC axis. The drive is attached to an SDR_Axis and the motor is commanded using the MC_XXX function blocks. Using the interface it is possible to launch the motor in jog mode, in position mode, to have it flip between two positions and to home the motor position.



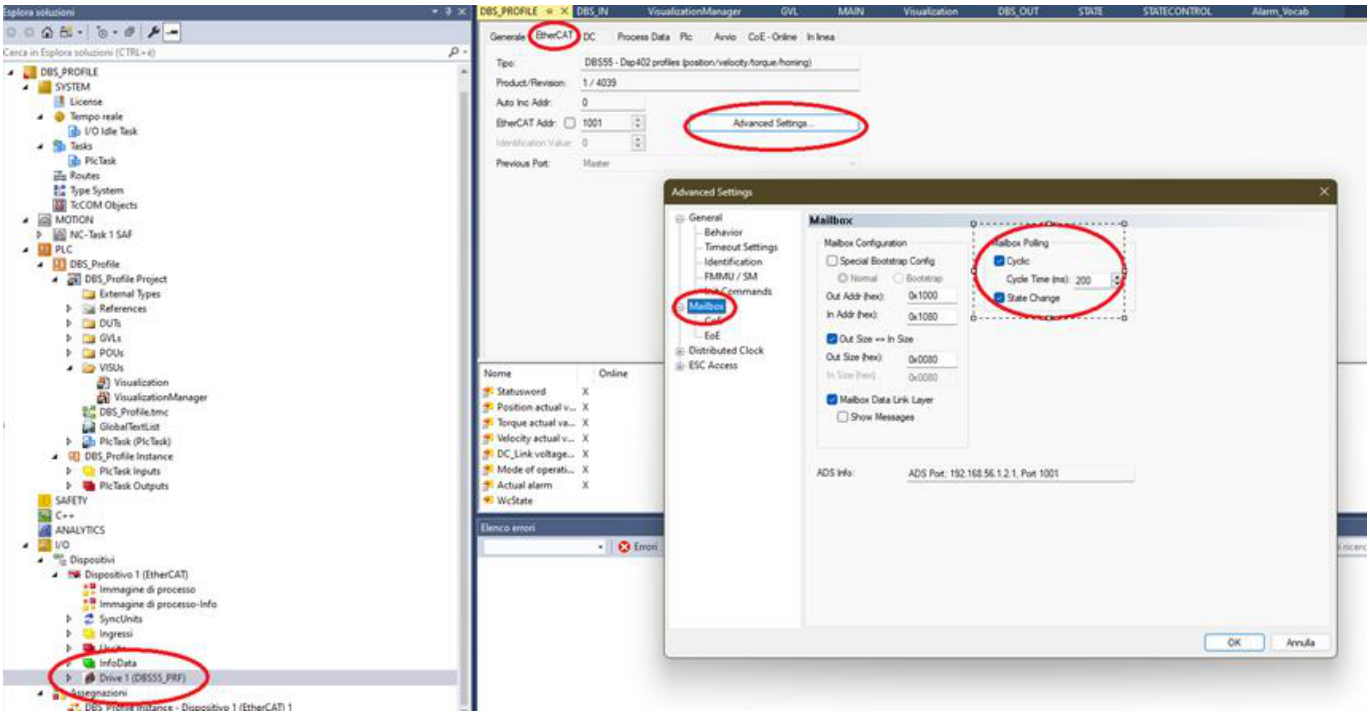
4.5. EoE Activation

If you want to activate the Ethernet over EtherCAT (EoE) feature on Beckhoff, follow this procedure:

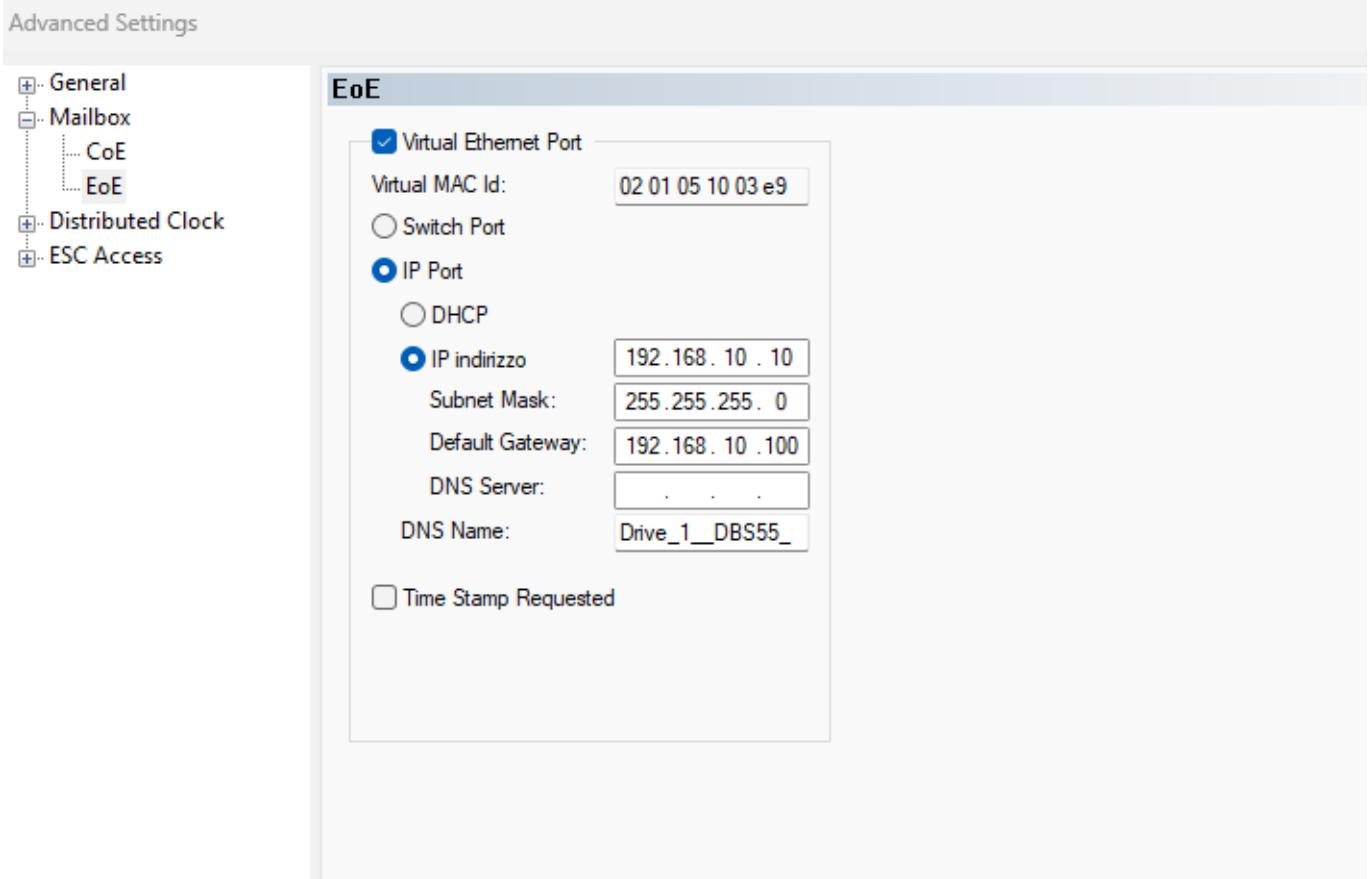
- Select the EtherCAT Master → EtherCAT tab → Advanced Settings → EoE Support → Enable.



- Select each device → EtherCAT tab → Advanced Settings → Mailbox → Activate Cyclic polling (50-100 ms).



- Then on the Device → EtherCAT → Advanced Settings → Mailbox → EoE tab, select and choose the IP address you want for it.



- Build the program, download it and it will work.

> Sometimes Windows can get complicated with routing. In that case check how things are set with the route print command in the Windows CMD; you could add a route with route add [EoE device IP address] [PLC IP address].