How to control a FAULHABER MC V3.0 ET out of a CODESYS environment

Summary

This application note describes the necessary steps to control a FAULHABER MC V 3.0 ET version using a CODESYS based PLC. The MC is connected via its EtherCAT port to the PLC.

Applies To

MC 5005 S ET, MC 5010 S ET, MC 5004 P ET and

MCS ET

Licensing

EtherCAT is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

Related FAULHABER Documents

Document	Description
Motion Manager 6	Instruction Manual for FAULHABER Motion Manager PC software
Quick start description	Description of the first steps for commissioning and operation of FAULHABER Motion Controllers
Drive functions	Description the operating modes and functions of the drive
Com Manual EtherCAT	Description of the EtherCAT services implemented in a FAULHABER MotionController

Description

The example shows a MC 5005 S ET operated in profile position mode controlled by a CODESYS PLC. The PLC application will start or stop the drive and does alternating position steps between two fixed absolute references while modifying the profile parameters in each step. It is created in structured text (ST).

In this example a Raspberry PI (RasPi) is used as a PLC runtime environment. As of now the runtime for the RaspPi can be downloaded free of charge out of the 3S Web shop (registration required) and can be used for training and testing purpose. Restrictions apply however. The maximum on time of the runtime is 2h for the free version. There is no limit for the number of starts however. And of course the RasPi is non hard real time environment, so a jitter in the timing is to be expected. Nevertheless this is a convenient low cost way to test the capabilities of such a

combination which has been successfully tested with distributed clock synchronization down to 2ms communication cycle time.

The example has been implemented and tested for either a RasPi 2B or a 3B version.

Figure 1 Test setup: Controller MC 5005 S ET + Motor 2250 BX4 + PLC RasPi 3B

The single Ethernet port of the RasPi has to be used for the EtherCAT interface. The access from the engineering environment installed at a PC to the runtime at the RasPi requires a second network interface. Therefore the RasPi has to be connected to the office network via either its onboard WLAN (RasPi 3B) or a USB WLAN dongle (RasPi 2B).

Following the installation of the RasPi, the CODESYS engineering environment and the CODESYS runtime are explained. These steps usually only have to be taken once at the very beginning. In a second step the creation of an application is explained and an example application is listed.

Software Installation

Before stepping through the example here, some installations are required.

Configuration and first tests of the FAULHABER MotionController are done using the FAULHABER MotionManager. The most recent version of the MotionManager can be downloaded directly from the FAULHABER web page.

The CODESYS environment has to be downloaded from the 3S Webshop. All the mentioned components are available free of charge. A registration for the web shop is required though.

Package	Source	Size
Setup_CODESYSV35SP10Patch2.exe (CODESYS Development System)	3S Web shop	1 GB
CODESYS Control for Raspberry PI 3.5.10.20.package	3S Web shop	10 MB
OSCATBasic.package	3S Web shop	8 MB
raspberrypi_codesysv3_firststeps_xx.pdf	3S Web shop	1 MB
NOOBS	Raspberry.org	1.5 GB

Table 1 List of software components required for a CODESYS environment

Install and configure the RasPi

Start with the installation of the NOOBS at the RasPi and configure it to at least have SSH enabled. VNC can be used to remotely connect to the RasPi sparing the extra monitor and keyboard for the PLC device.

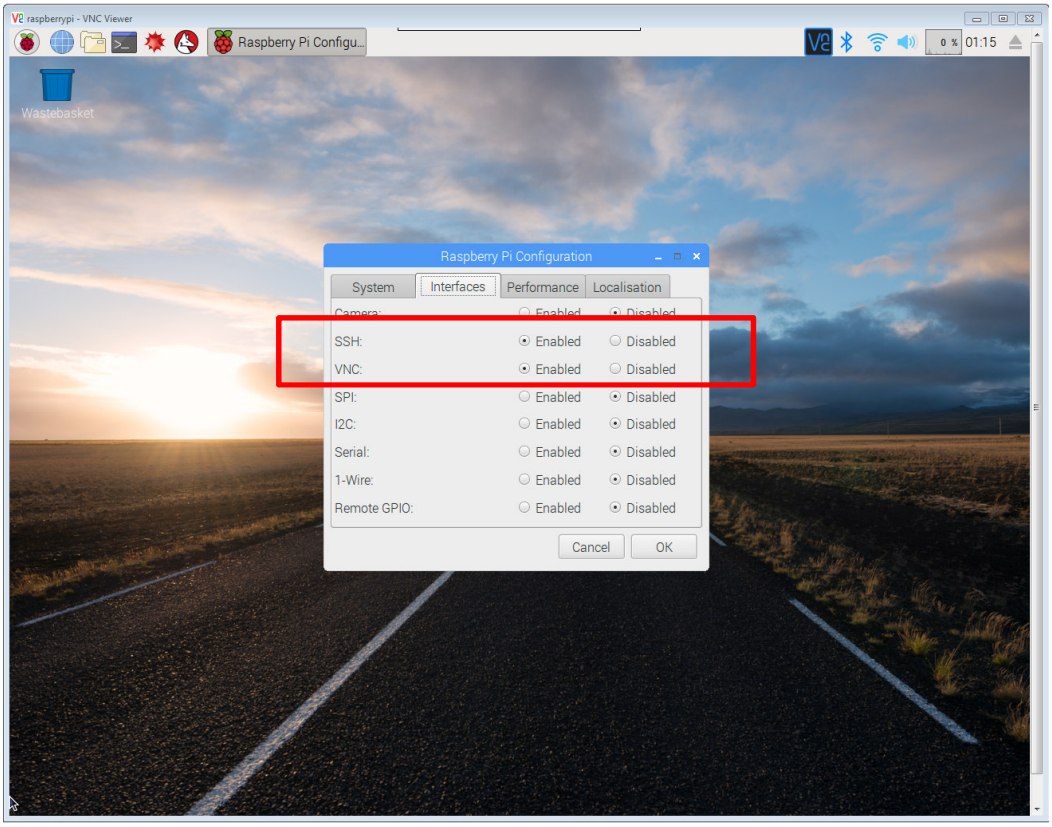
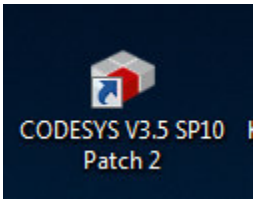


Figure 2 Raspberry Startup screen showing the system preferences. Additional connection e.g. SPI might be enabled, if additional hardware is connected to the RasPi

SSH is used by the CODESYS engineering environment to install its runtime.

Install the CODESYS Environment



In a second step install the CODESYS Engineering Environment to your PC. Here we used the V3.5 SP10. This is the main tool for creation of PLC programs and visualizations. Installation will take some time and be prepared to have additional components downloaded and installed to your computer in this step (see Figure 3).

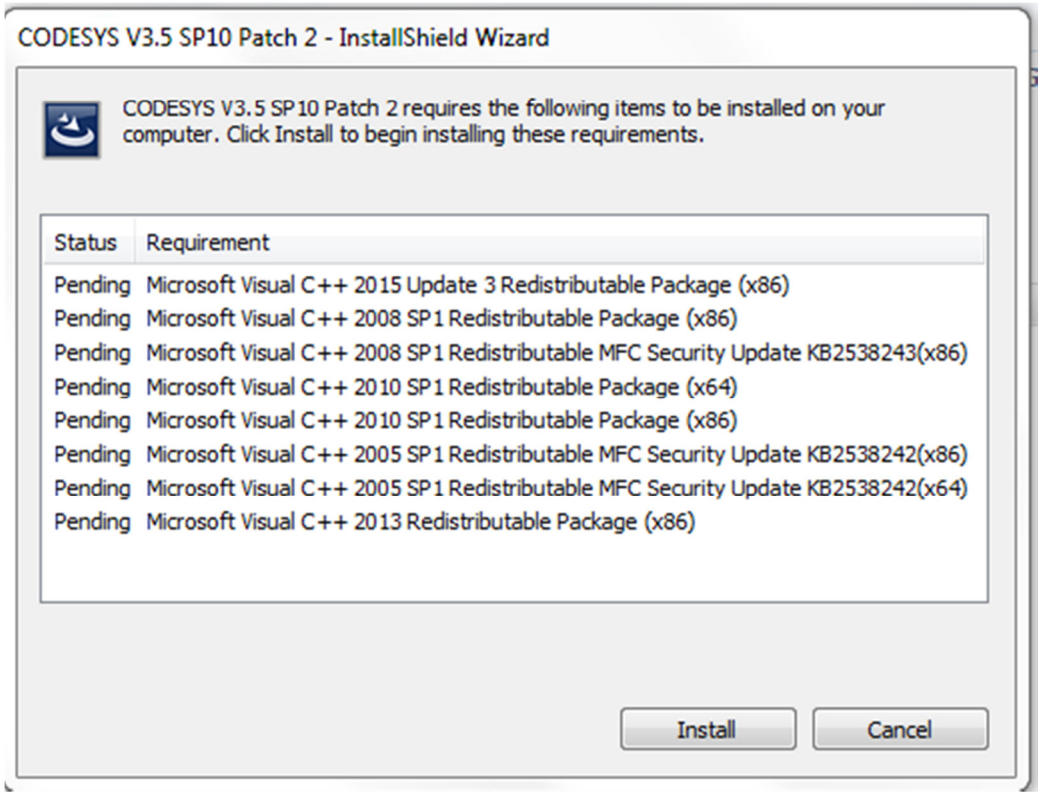


Figure 3 Additional components installed by the CODESYS engineering environment

After the installation of the engineering environment, we can add the specific components for the example here.

First of all we need to download the RasPi Solution from the 3S web shop. This solution is distributed as a package that needs to be added to the engineering environment via the package manager in the Tools menu (see Figure 4).

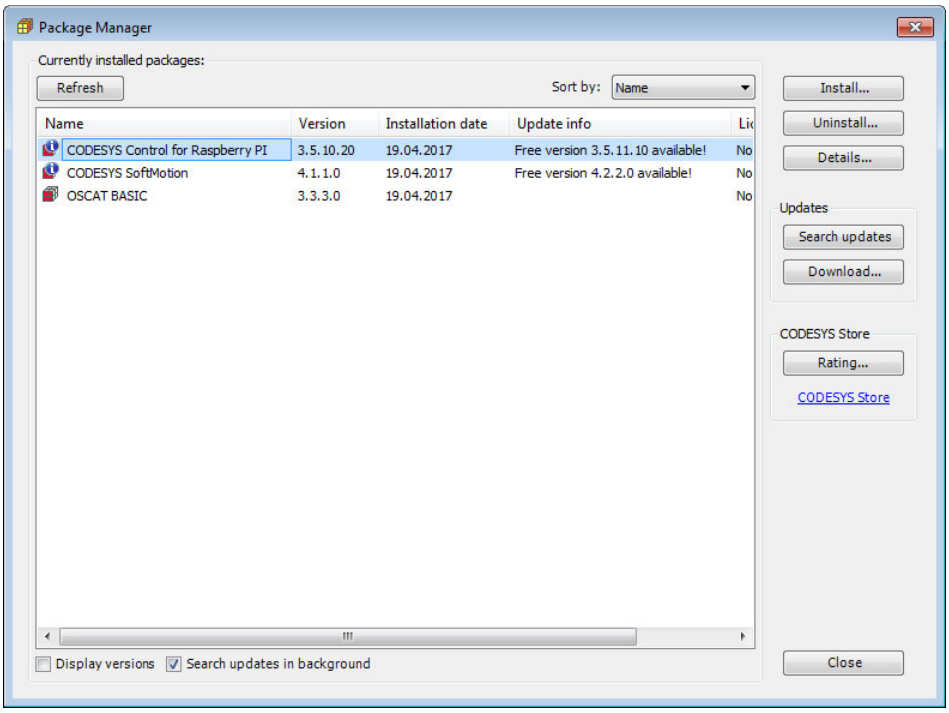


Figure 4 CODESYS Package manager allows to add additional components such as the RasPi environment

You might also download and add the OSCAT Basic lib, a library with general purpose function blocks for the CODESYS environment. We do use one of them later in the listing.

The lib can then be installed using the Tools/LibraryRepository.

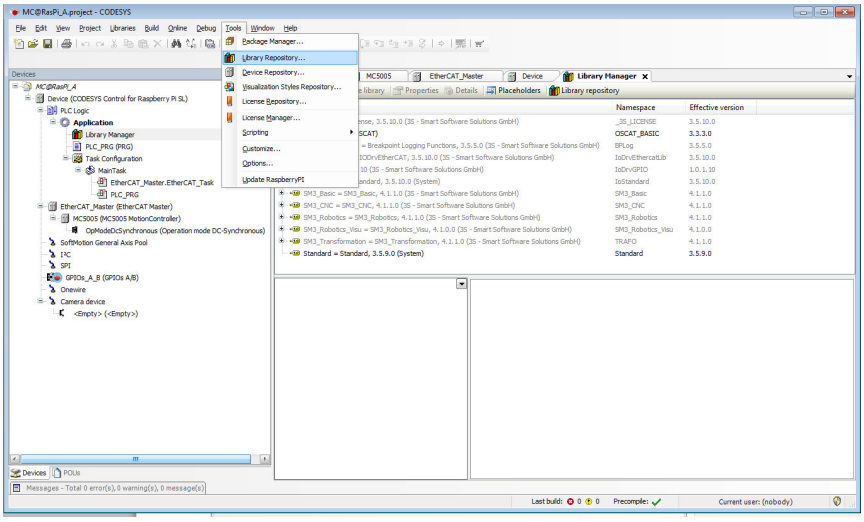
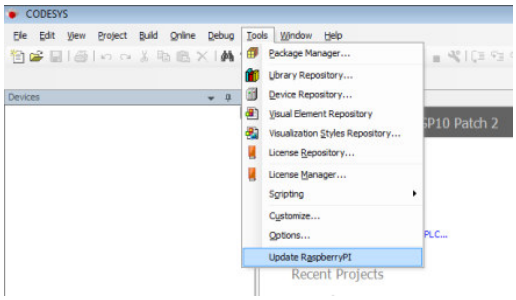
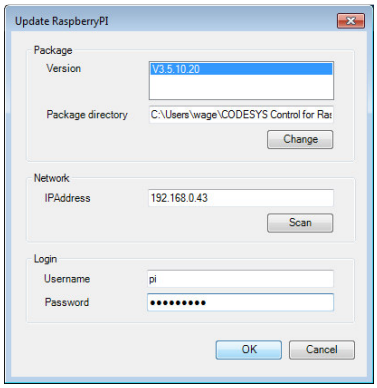


Figure 5 Installation of additional libraries using the LibraryRepository

Only after this step, we are now able to install the CODESYS runtime component at the RasPi. This installation is done directly out of the CODESYS runtime environment. There is an extra menu entry in the Tools menu Update Raspberry). This will open up a window where you have to identify your RasPi (IPAddress) and add the login information.





Press Scan to identify the RasPis is your network environment. Alternatively add the IPAddress manually.

Add the login information or the RasPis SSH. Default is:

Username: pi

Password: raspberry

Figure 6 Update Raspberry

Finally, in order to use the FAULHABER MCs in CODESYS projects, the device description files distributed with the MotionManager have to be installed to the CODESYS Device Repository – once again the Tool menu. The description files can be found in the installation directory of the MotionManager/ESI, e.g. something like C:\Program Files (x86)\Faulhaber\Motion Manager 6\ESI.

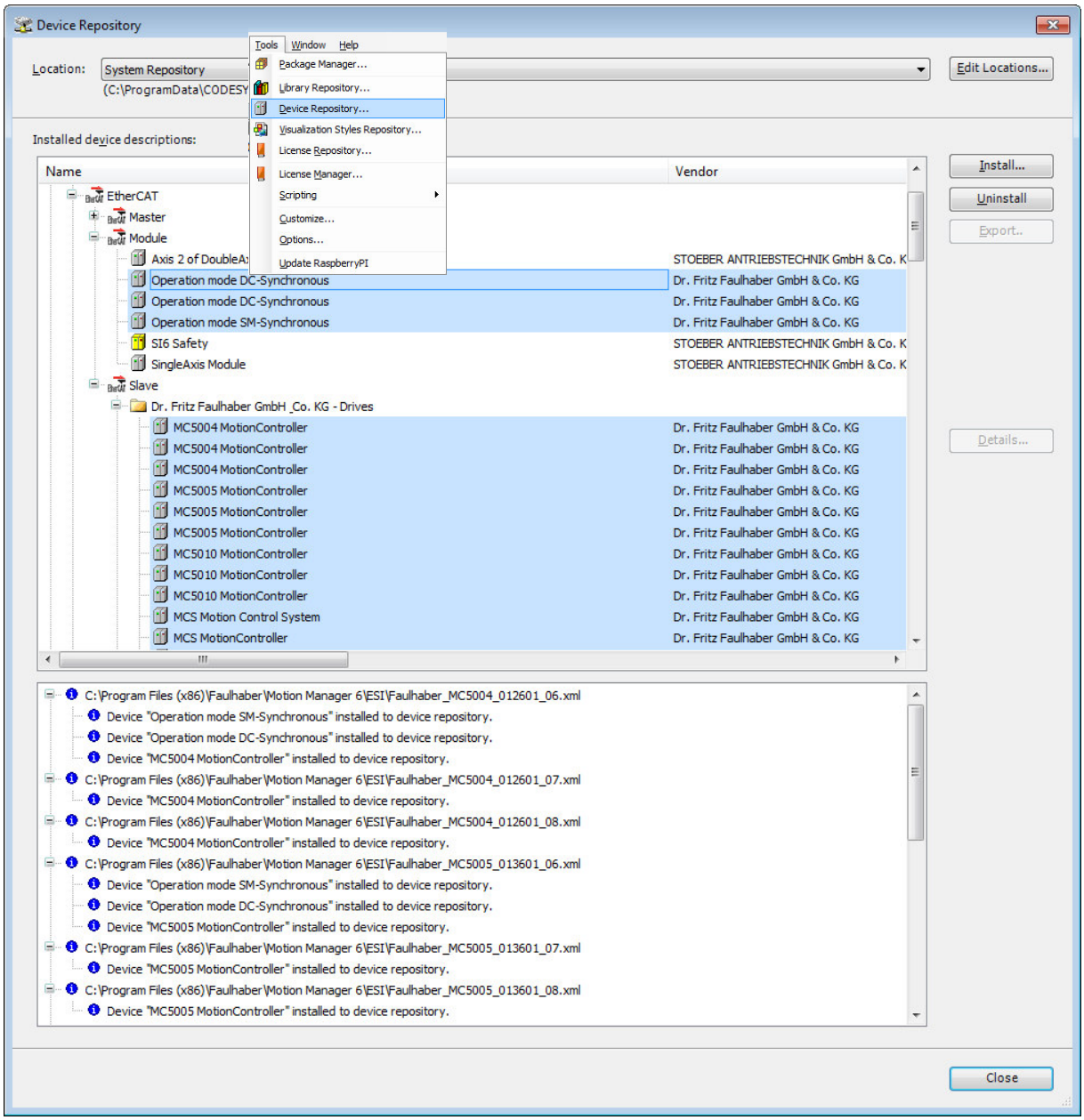


Figure 7 Adding FAULHABER MotionController to the CODESYS Device Repository

After this final step all the prerequisites are finished. We can now start with the application example.

Create a new PLC application

Creation of a PLC application requires a minimum of 4 steps.

1. Open a new PLC project
2. Create a system description
3. Add the functionality
4. Test the system behavior

Open a new project

A new CODESYS project can be created using either the direct link in the Start Page or via menu File/New Project.

Here we use a Standard project. Select whatever path is convenient and enter a project name.

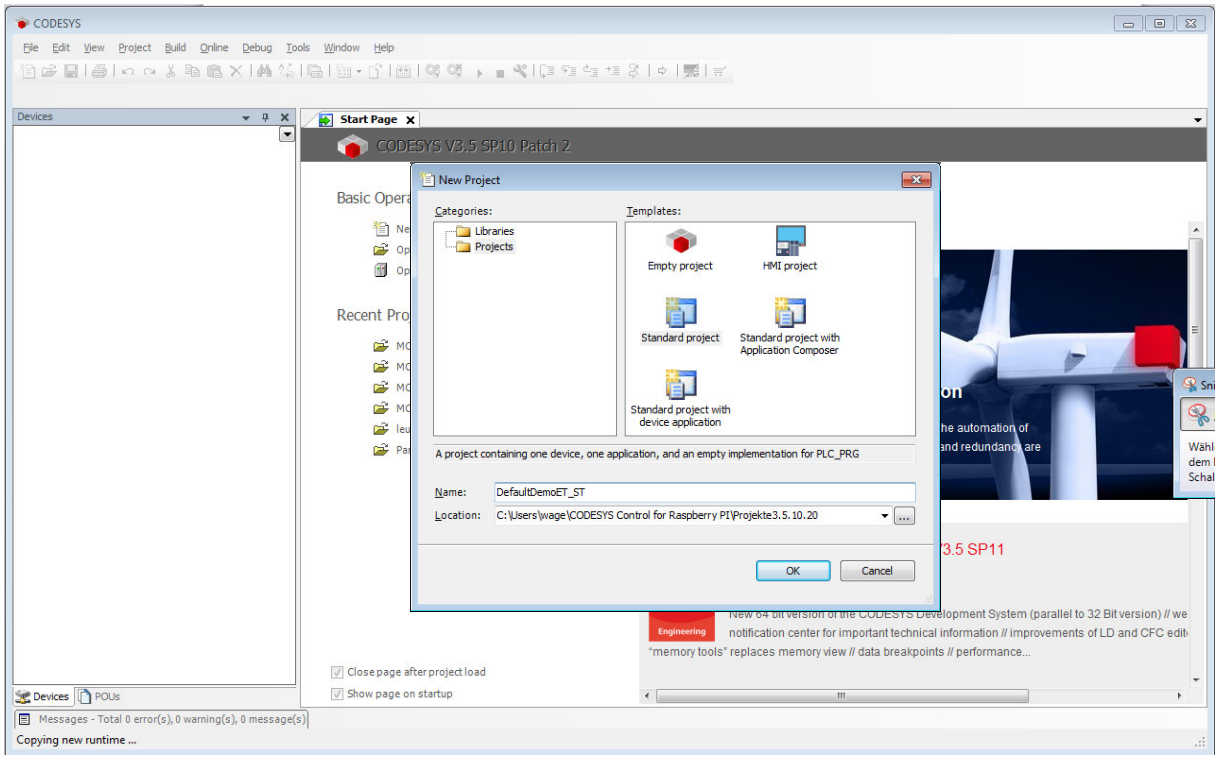


Figure 8 Open an new CODESYS project

In a second step the target type and programming type for the single program organization unit (POU) PLC_PRG are selected (see Figure 9).PLC_PRG is part of the new project and already configured to be executed by the main task.

Here we choose structured text (ST), a text based coding very similar to PASCAL.

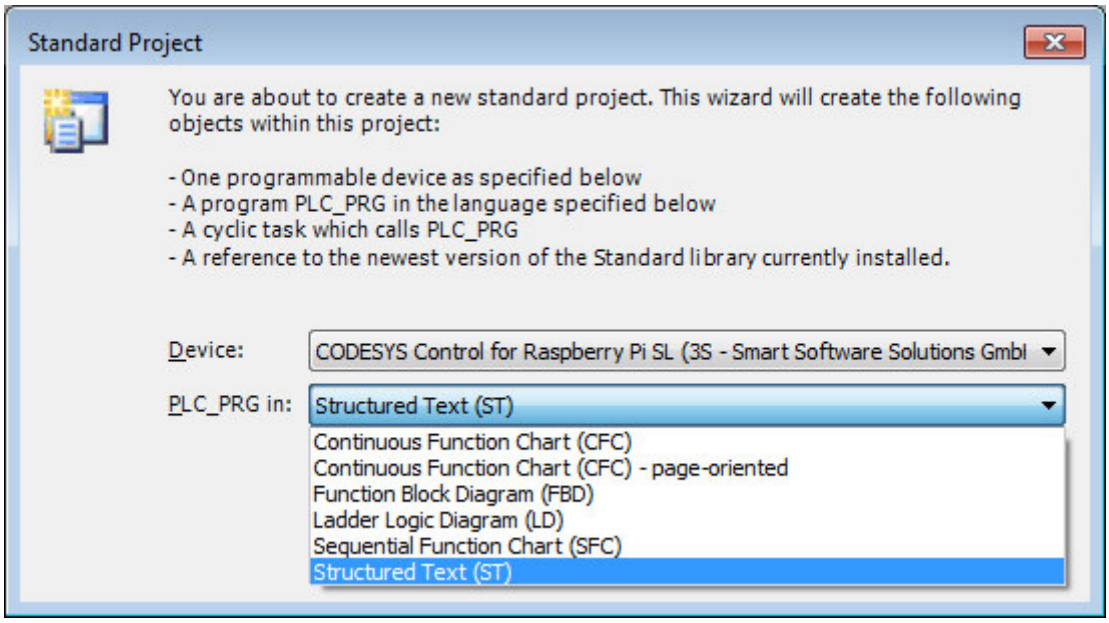


Figure 9 Selection of the target runtime and the programing style

Finally the new project is displayed in the project tree (see Figure 11). Listed under the Device node (which represents the RasPi PLC controller) there is one node for the PLC_Logic. All other ones are empty.

Within the 3S startup manual for the RasPi solution several simple examples in combination with different types of I/O connected to the RasPi are explained. Here we will focus on a simple system out of the RasPi and a single FAULHABER MC 50xx connected via EtherCAT (Figure 1).

In our example the Ethernet port of the RasPi is directly connected to the EtherCAT port labeled IN at the MotionController. No additional connection is necessary.

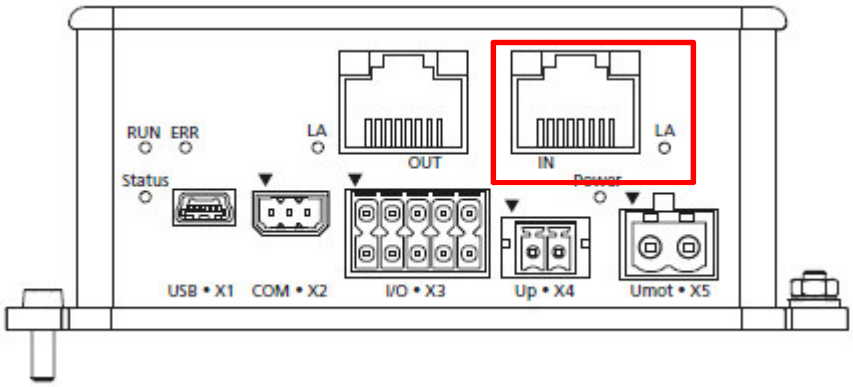


Figure 10 Connections of a FAULHABER MotionController MC5005/10 S ET

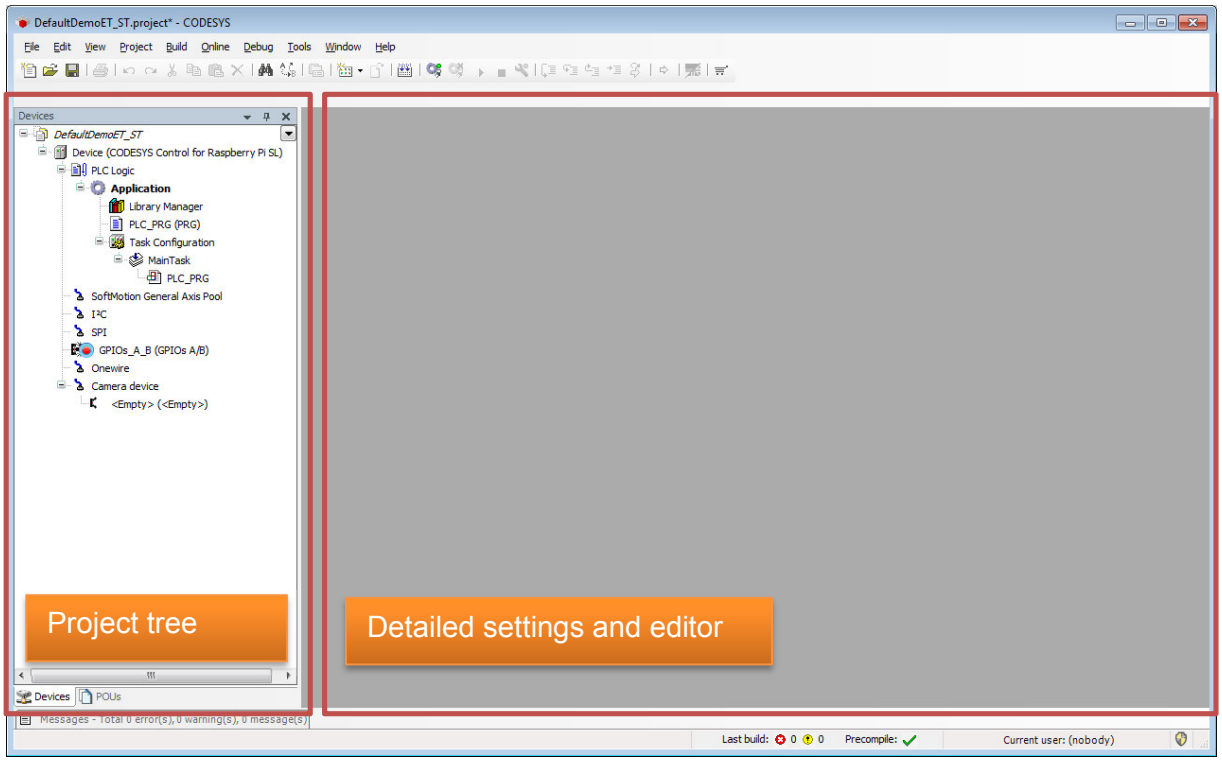


Figure 11 Project view in the CODESYS environment

Connect to the Device

The CODESYS Development System can generate code for different targets. In order to tailor the configuration to the selected target your RasPi should be added to the project in a first step. To do this, a double-click on the device node in the project tree will open the device page (Figure 12) from where the network can be scanned for devices. Several CODESYS runtime systems might be identified. RasPi based targets cannot be identified by a blinking LED or something similar. One of the properties of the identified devices is the Device_Name. So if you plan to use several of them, you should use different names here[1].

The settings of the connected device are displayed on the device page after this step only (see Figure 13).

[1] Targets can be renamed using the Device menu in the Device settings page.

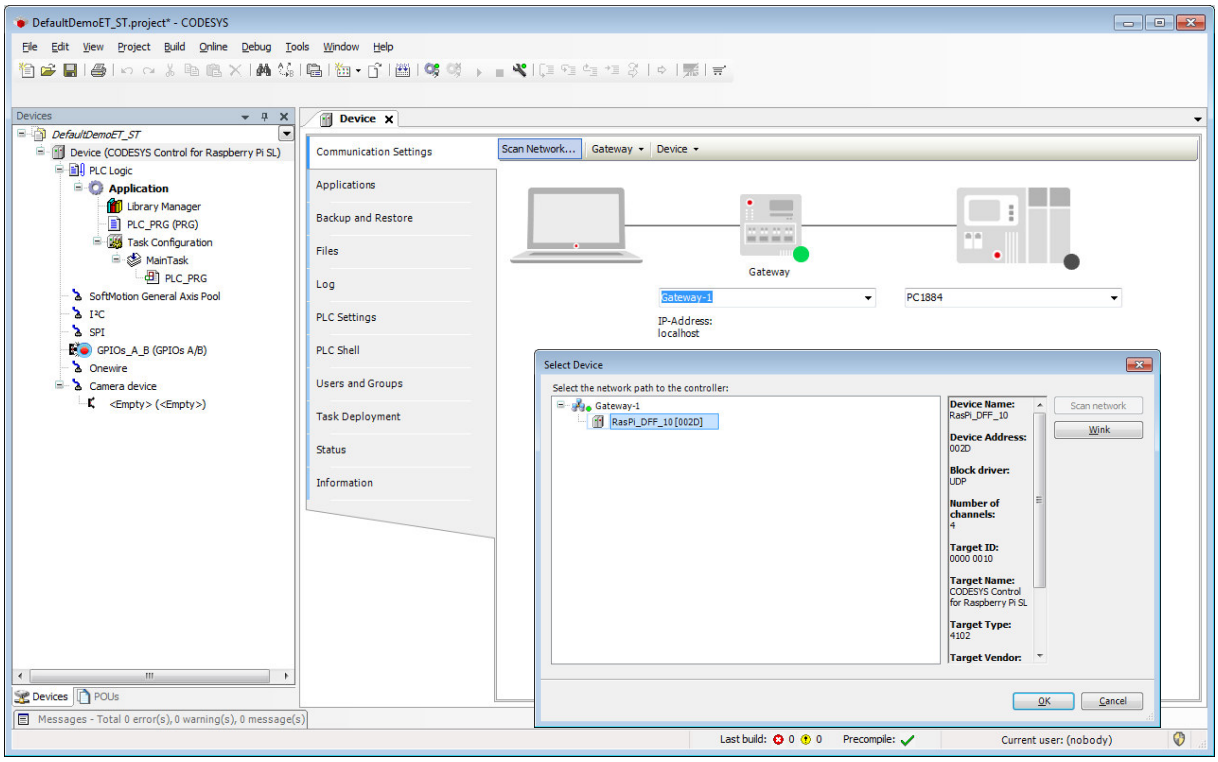


Figure 12 Identify the target controller (RasPi)

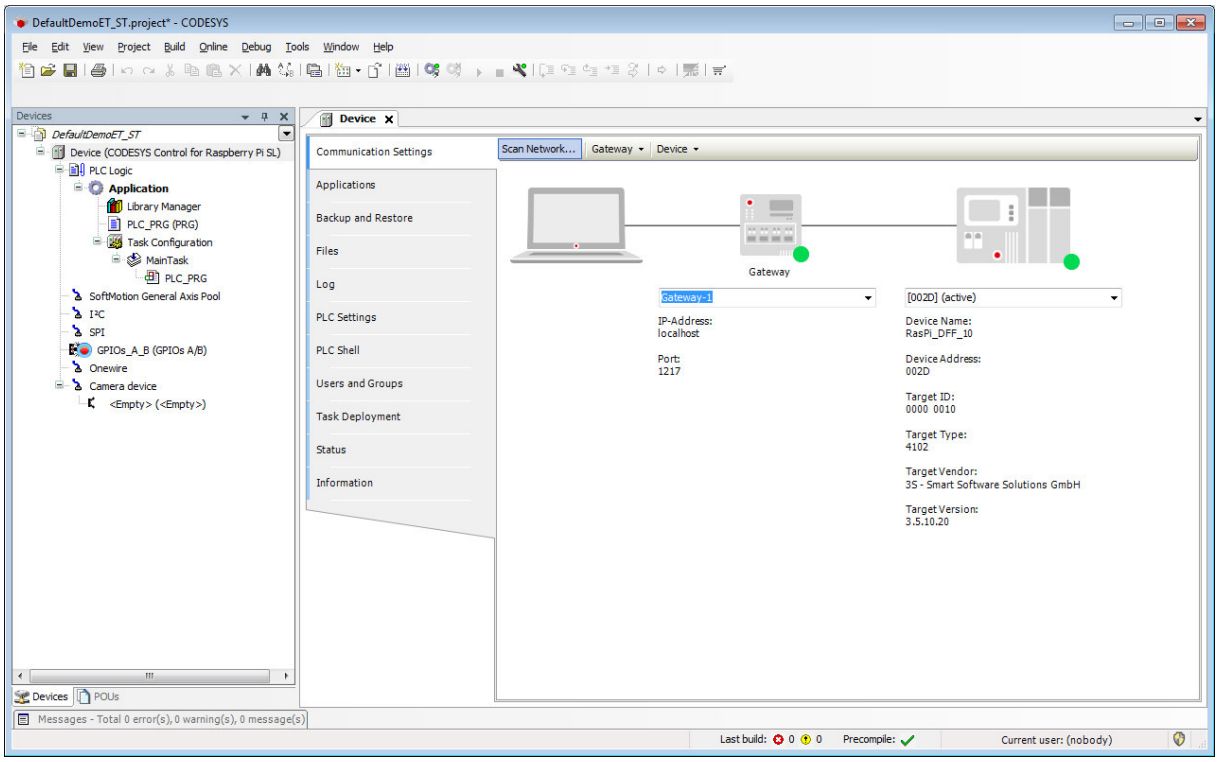


Figure 13 Project view with identified target controller

Create the system definition offline

Before we start programming the application, we should create a complete system definition. Here we need to add the EtherCAT subsystem. Additional examples of how to add drivers for SPI or I²C based subsystems can be found directly in the 3S startup description.

Additional components can be added to the project using the context menu in the project tree. Here the master is added to the node Device using Add Device. First we need to add the driver for the EtherCAT – the EtherCAT Master - using add Device out of the context menu of the project. Available devices can be selected from the Add Device window using filters and the tree view. As of now, there is only one EtherCAT master, so this is an easy one.

A double-click on the tree node of the EtherCAT master will open the settings of the component (Figure 15).

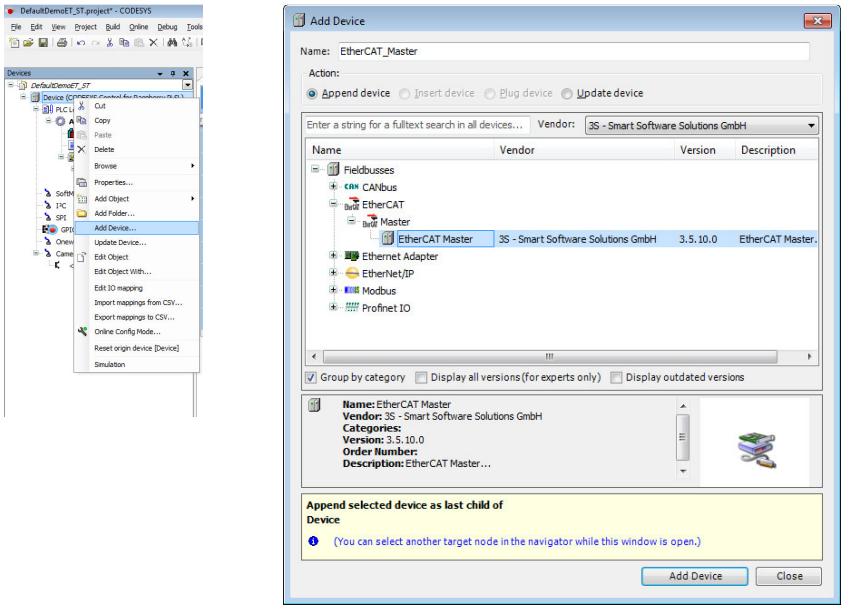


Figure 14 Add the EtherCAT Master to the project using Add Device

Next we need to configure the network interface to be used by the EtherCAT master at the target controller. A list of available network interfaces can be displayed by clicking the Browse … button (Figure 15). Here we have to select the single Ethernet port of the target: eth0. The identification within the project can be either MAC based or name based. The latter allowing for easier portability of the project.

The cycle time of the EtherCAT system defaults to the cycle time of the main task. For the RasPi this is 4ms. We did test cycle times down to 2ms successfully with either a RasPi 2B or 3B target[2].

[2] If a different timing is required, first change the cyclic time of the main task, then adjust the Cycle Time of the EtherCAT and re-compile / re-load the system.

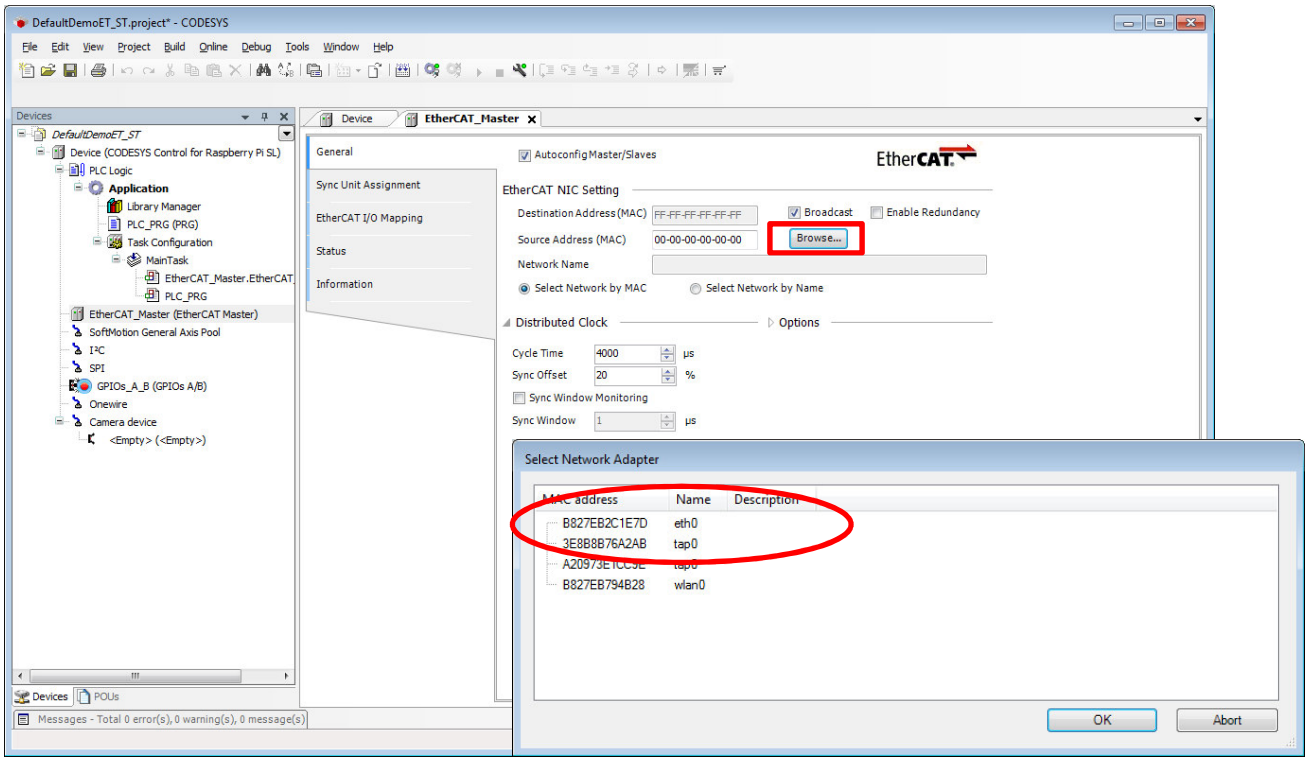


Figure 15 EtherCAT Master settings and list of available network adapters

After the basic configuration of the EtherCAT subsystem we can add slaves to the system. This can either be done offline, by manually adding devices to the EtherCAT bus node or using the online Config Mode  being available either from the context menu of the device in the project tree or out of the menu. Online Config will start the EtherCAT system configured in the last step and will scan the bus for known devices. Online Config avoids picking the wrong device or device revision.

In offline mode, you need to add the devices manually using the Add Device entry of the Ether-CAT master context menu. When using offline mode, please verify to have the correct device types and revision added to your project (Figure 16).

By default Vendor ID and product ID of the nodes are checked during startup of the EtherCAT subsystem. So even if a MC 5010 S ET and a MC 5005 S ET look very similar, they must not be mixed up here. Product revision is not checked by default, however in order to avoid communication problems the correct device description should be selected in offline mode.

Online mode even allows updating the contents of the EtherCAT slave EEPROM. This has to be done manually after a firmware update of the MC[3].

[3] A firmware update may add objects to the object dictionary of a servo drive. In order to correctly access all objects, the correct device description has to be used within the project. The device revision of an EtherCAT drive is identified via the device revision entry in the slave EEPROM.

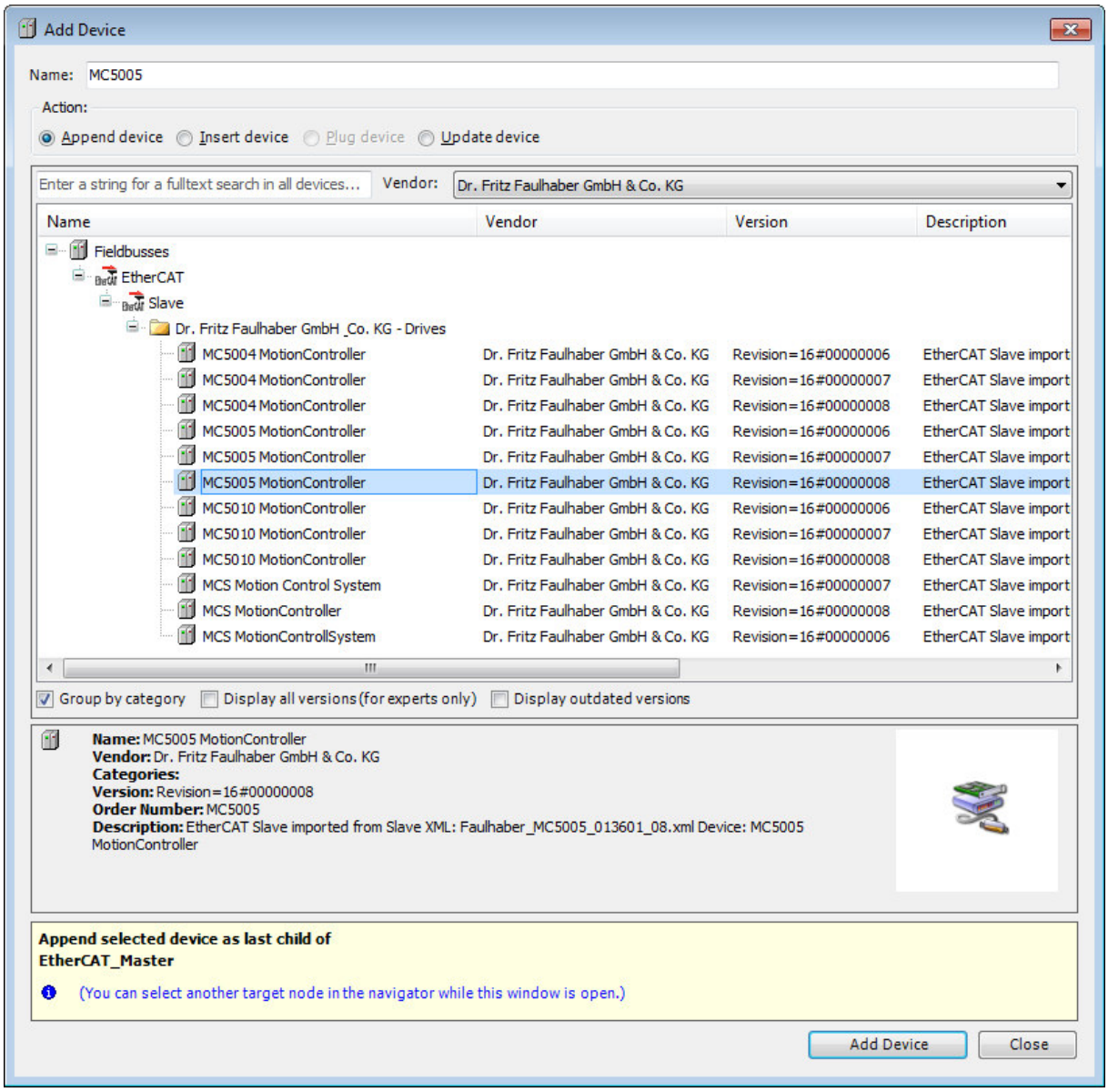


Figure 16 List of known devices

The configuration of the EtherCAT slave is done entirely using the slave settings of the MC node in the CODESYS Development System (see Figure 17). No configuration via FAULHABER MotionManager is necessary.

Review and modify the settings of the connected MC 5005 by double-click on the node MC5005 in the project tree an enable the expert settings. Expert settings are necessary to modify the PDO mappings of the device.

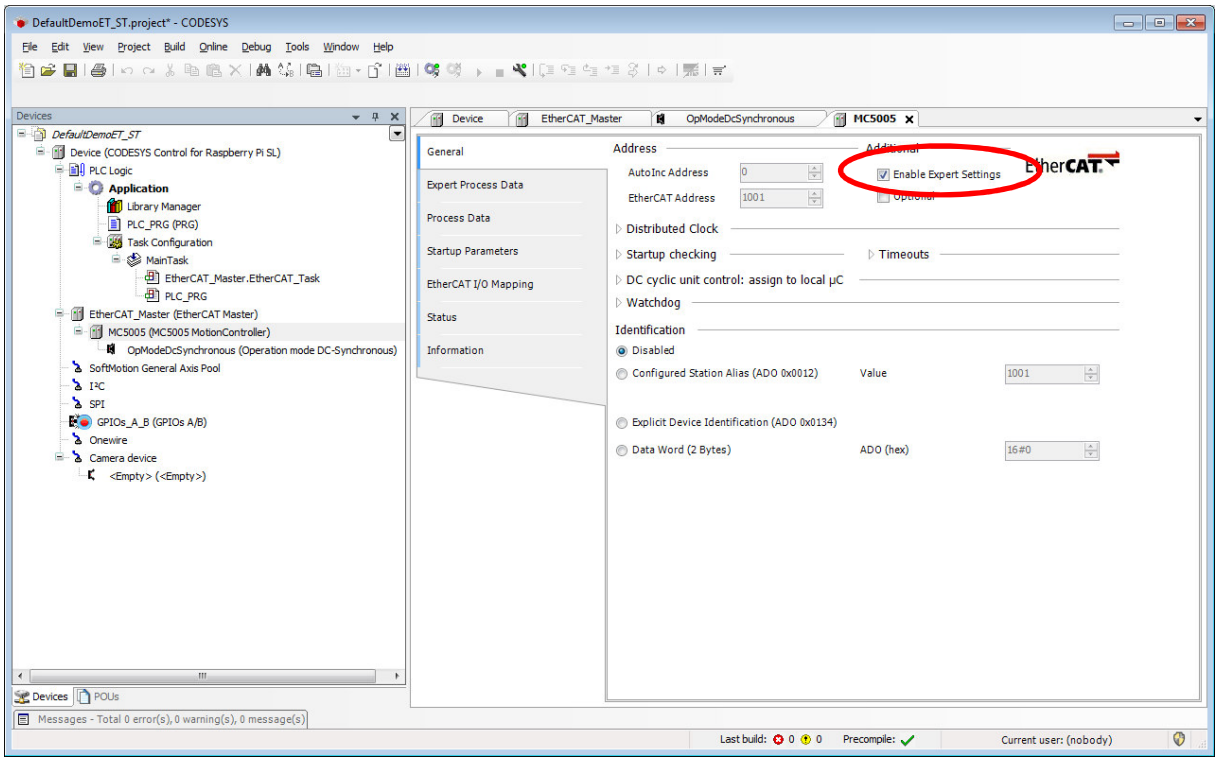


Figure 17 Basic EtherCAT Slave settings of the CODESYS system – expert settings enabled

By default Distributed Clock is used as the synchronization method for the FAULHABER drives. Synch type selection is done based on a module concept. The synch type used in the application is interpreted like being a module plugged into the device. The selected synchronization type is therefore visible as a module plugged into the MC 5005 in Figure 17. To enable the Distributed Clock at the MC please check the entries at the General Tab according to Figure 18.

If Synchmanager (SM) synchronous operation is to be used, use the context menu of the MC 5005-OpMode in the project tree to change the plugged OpMode.

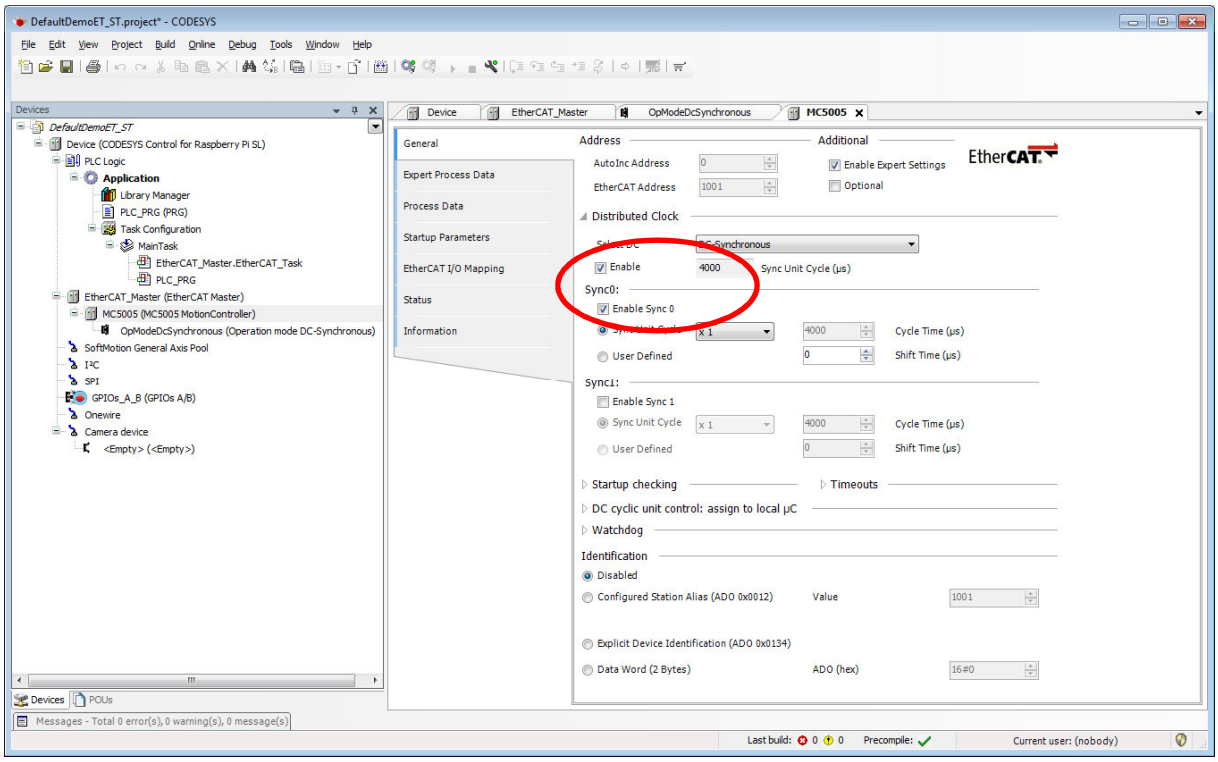


Figure 18 Synch settings of the EtherCAT slave

Create / Modify PDO Mapping

In order to tailor the data exchange between the PLC application and the MotionController to the application we need to review and configure the process data objects (PDO) of the drive using the Expert Process Data Tab of the drive settings.

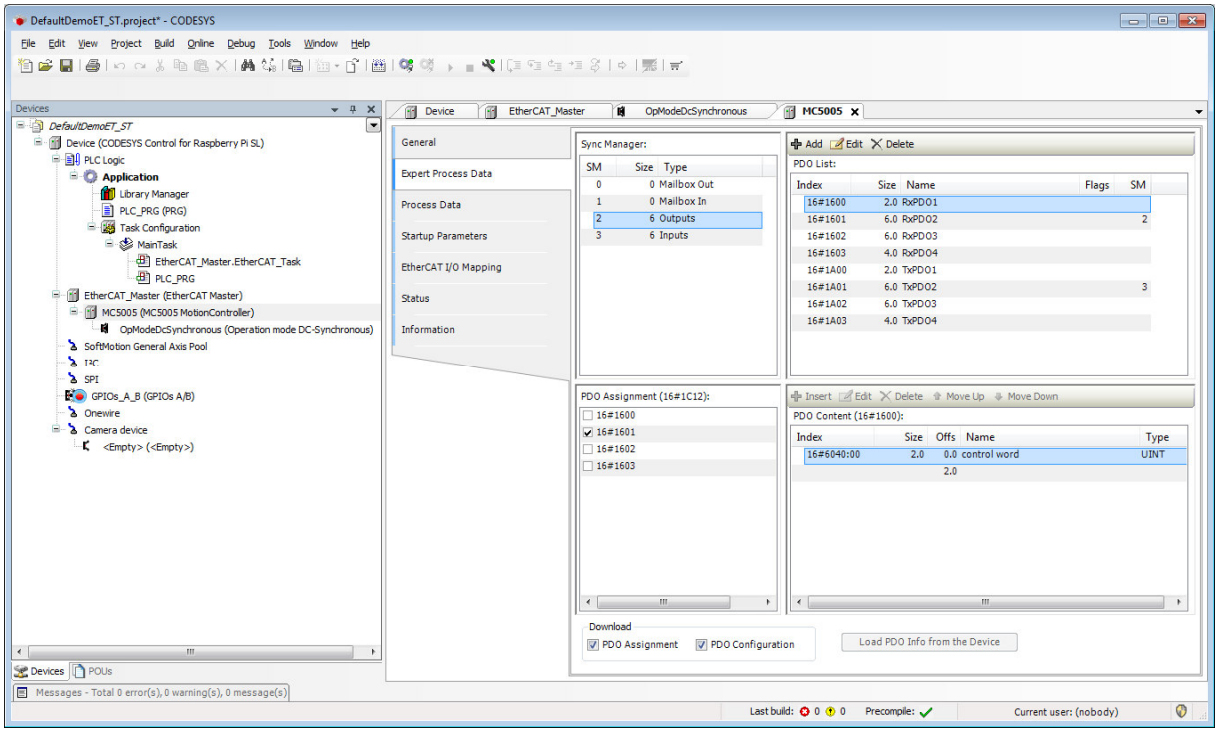


Figure 19 Expert view of the PDOs (defaults)

A maximum of 4 objects can be mapped to a single PDO.

To maintain the correct order of processing the controlword (0x6040.00), the OpMode (0x6060.00) and the target position (0x607A.00) have to be mapped to a single PDO. The controlword must not be mapped to more than 1 active PDO.

Within the default mapping the controlword is mapped to all RxPDOs, RxPDO 2 – 4 each covering a different simple access of controlword and the target value for either pos control (RxPDO2), speed control (RxPDO3) or torque control (RxPDO4). Same for the Tx direction. To avoid a mixed update of the controlword, only PDO2 is activated by default.

Here we want to access:

Read		Write	
0x6041.00	statusword	0x6040.00	controlword
0x6061.00	Modes Of Op. Display	0x6060.00	Modes of Operation
0x6064.00	Position Actual Value	0x607A.00	Target Position
0x606C.00	Velocity Actual Value	0x6081.00	Profile Velocity
0x6077.00	Torque Actual Value	0x6083.00	Profile Acceleration
0x2311.01	Dig Input logic state	0x6084.00	Profile Deceleration

Table 2 List of all parameters used in the interaction between PLC and servo drive

So the PDO Mapping here might be:

Read from the MC	Write to the MC
TxPDO1: not used, no change necessary	RxPDO1: not used, no change necessary
TxPDO2: 1. 0x6041.00 (statusword) 2. 0x6061.00 (Modes of Op Display) 3. 0x2311.01 (Dig Input logic state) 4. -	RxPDO2: 1. 0x6040.00 (controlword) 2. 0x6060.00 (Modes of Operation) 3. 0x607A.00 (Target Position) 4. –
TxPDO3: 1. 0x6064.00 (Position actual value) 2. 0x606C.00 (Velocity actual value) 3. 0x6077.00 (Torque actual value) 4. -	RxPDO3: 1. 0x6081.00 (Profile Velocity) 2. 0x6083.00 (Profile Acceleration) 3. 0x6084.00 (Profile Deceleration) 4. -
TxPDO4: not used, no change necessary	RxPDO4: not used, no change necessary

Table 3 Proposed PDO mapping

To edit the mappings the 4 tables in the Expert Process Data tab can be used.

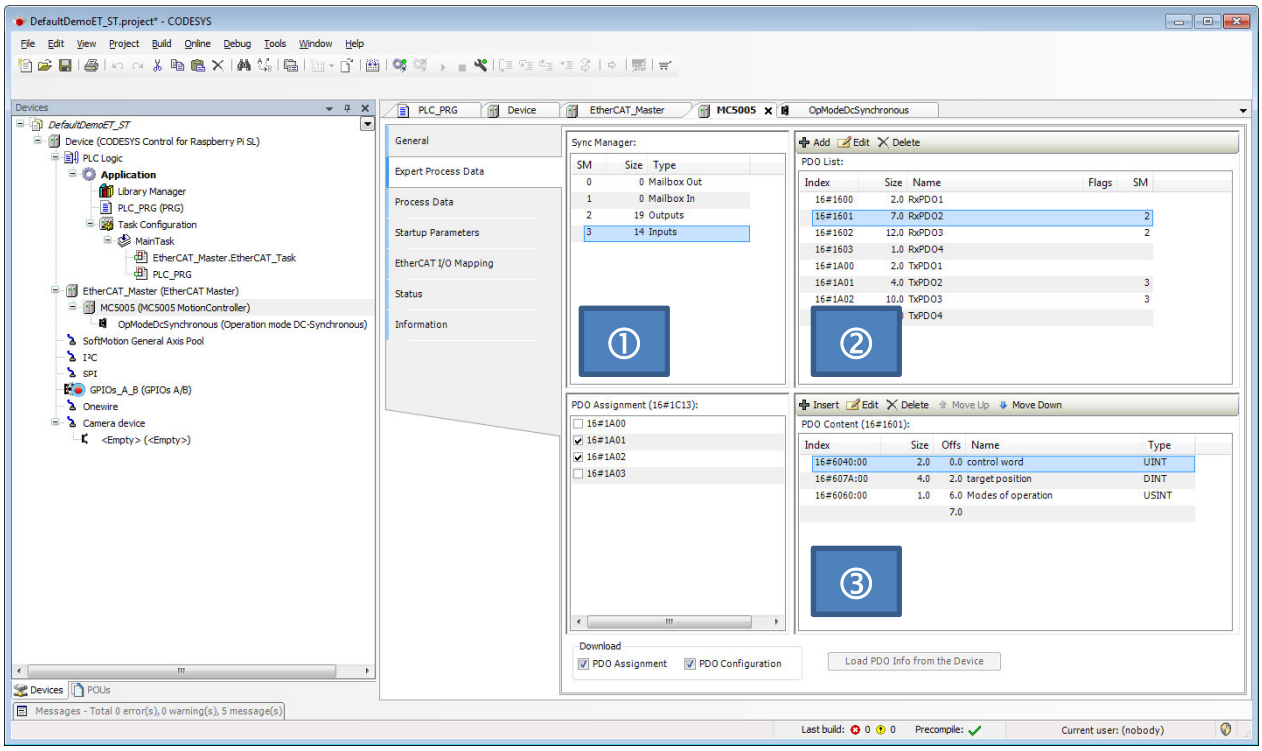


Figure 20 Expert view of the device PDOs, adjusted to Table 3

Edit the RxPDOs:

1. Select the Outputs in the Synch Manager table and enable RxPDO3 by checking it in the PDO Assignment
2. Select RxPDO2 in the PDO List
3. Add a new entry to RxPDO2 in the PDO Content table using the ✚Insert. Here we add 0x6060.00 Modes of Operation to the mapped entries.
4. If necessary the order of the objects can be modified using the Move up or Move down buttons
5. Select RxPDO3 in the PDO List
6. Delete the default entries (controlword, target velocity) using the ✖Delete and add 0x6081.00 - Profile Velocity, 0x6083.00 – Profile Acceleration and 0x6084.00 – Profile Deceleration.
7. Enable additional PDO3 by checking them in the PDO Assignment table

⇨ The total size of the Outputs in the Synch Manager table should now be 19 bytes

Edit the TxPDOs:

1. Select the Inputs in the Synch Manager table and enable TxPDO3 by checking it in the PDO Assignment
2. Select TxPDO2 in the PDO List
3. Add 0x6061.00 - Modes of Operation Display to the mapped entries
4. Add 0x2311.01 – Digital Input Logic States to the mapped entries
5. Delete 0x6064.00 – Position actual value from TxPDO2

6. If necessary the order of the objects can be modified using the Move up or Move down buttons
7. Select TxPDO3 in the PDO List
8. Delete the default entries of the 0x6041.00 statusword and add 0x6064.00 – Position actual value, 0x6077.00 – Torque actual value.
9. Enable additional PDO3 by checking them in the PDO Assignment table

⇨ The total size of the Inputs in the Synch Manager table should now be 14 bytes

There is no need to explicitly configure the mappings beforehand in the MC using FAULHABER MotionManager because the PDO assignment and mappings are downloaded to the MC by the PLC during initialization.

Different from the CANopen environment we don't have to care about the transmission type of PDOs they will be updated in every communication cycle anyway.

Additional configuration of the MC

To configure the correct motor or the I/Os of the MC please use FAULHABER MotionManager and the secondary interface of the MC which is either the USB interface for an external controller or the RS232 of an MCS. This configuration is mandatory before the motor is operated by the PLC.

Create a PLC program

By default the program sequence of the project is stored in the PLC_PRG component of the project tree. Here it is a structured text (ST) version. Structured text is similar to PASCAL. A double-click on the PLC_PRG entry of the tree opens the editor tab.

The editor is divided into the variables section and the code section. All variables and instances of function blocks have to be defined in the variables section in between the key words.

Definition of global variables

VAR

…

END_VAR

So if we want to interact with the MC we usually have to read the statusword and write the controlword. Both of them unsigned 16 bit.

The entries would look like:

```
VAR
    ControlWord: UINT:=0;
    StatusWord: UINT;
END_VAR
```

The variable ControlWord is set using a default value, StatusWord does not need a default value, it's updated out of the controller anyway.

Variables can also be created on the fly during coding. Each time a new variable name is used a window will pop up and gather the necessary attributes.

Map PLC variables to the process image

Please note: the listing above does only generate the two variables for the PLC program. In order to access the MC the two variables have to be linked to the PDOs. This can be done using the EtherCAT I/O Mapping tab of the MC node in the project tree. A double-click into one of the variables (empty or already connected) will allow to select a new variable.

So here we should map the entry controlword in RxPDO2 to the PLC variable ControlWord and the statusword entry in TxPDO2 to the PLC variable StatusWord:

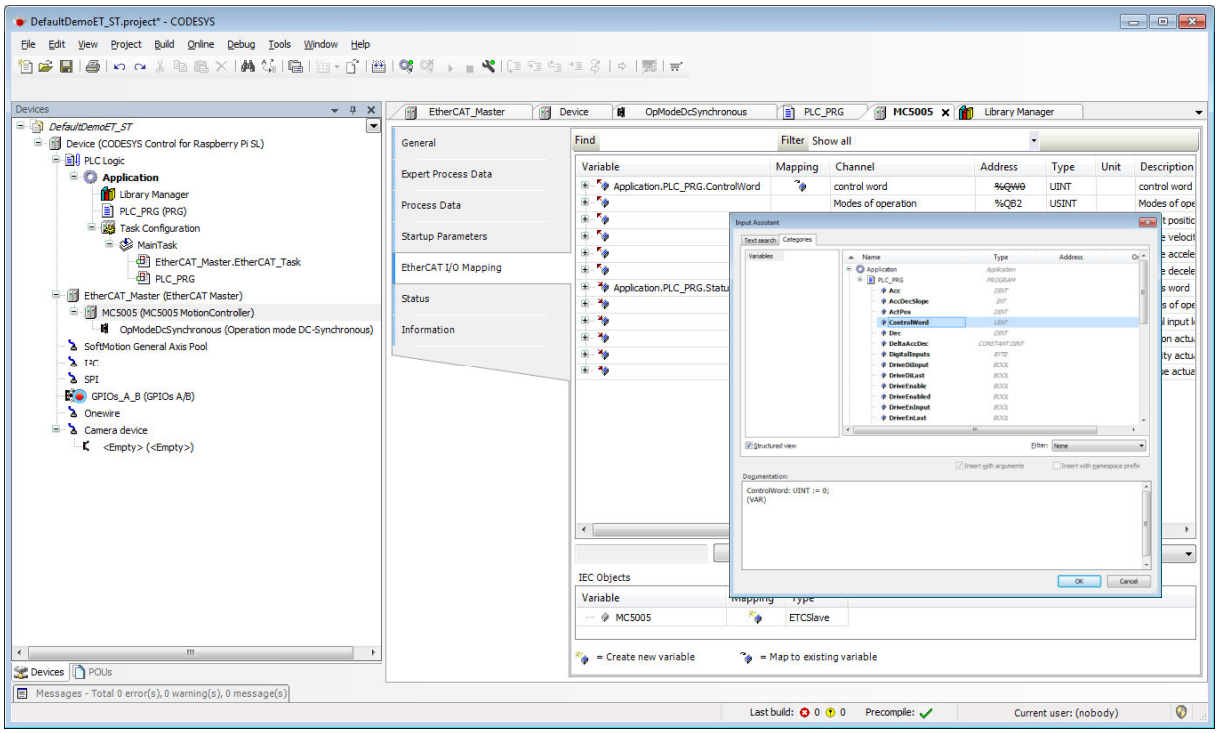


Figure 21 Mapping dialog - select the application variable to be updated by / to update the PDO

Access to a single bit within the ControlWord/StatusWord

Sometimes we need to access a single bit within a variable and want to ignore the others. This can be done e.g. for the flags TargetReached (bit 10) and SetPointAcknowledge (bit 12) of the PP mode using the .Bit operator. The result is a Boolean information.

```
//Flags of the PP mode
IsInPos := StatusWord.10;
SetPointAck := StatusWord.12;
```

Add additional libs to the project

If available it's recommended to rely on existing, proven code. This can be done using libraries. The OSCAT Basic lib installed earlier supplies some general purpose function blocks and should be added to the project. This is done using the Library Manager entry of the project tree. Once again a double-click will open the view where we can add a library. Filters are supported.

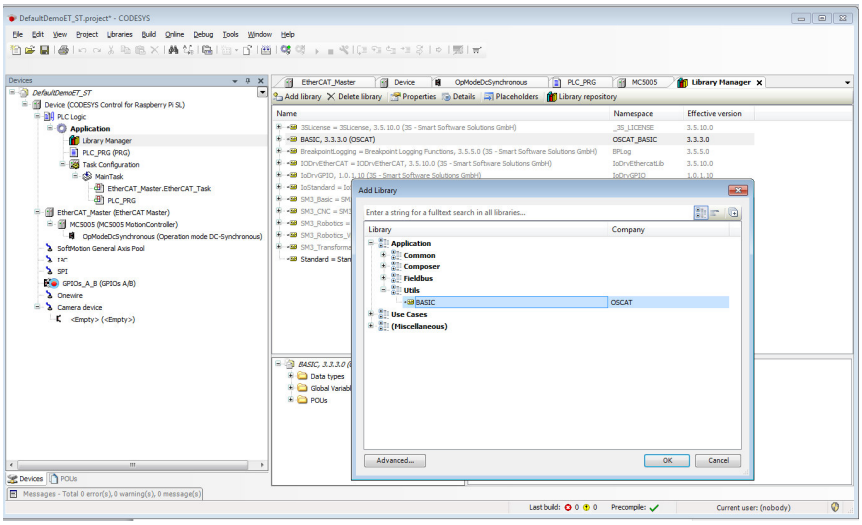


Figure 22 CODESYS LibraryManager

Download and start the application

After having entered the code, the application can be compiled using either the Build menu, the build-icon or F11.

A successfully built application can be downloaded to the PLC using Online/Login (or the login icon). The application is downloaded but will remain stopped until we explicitly start it using Debug/Start or the start icon.

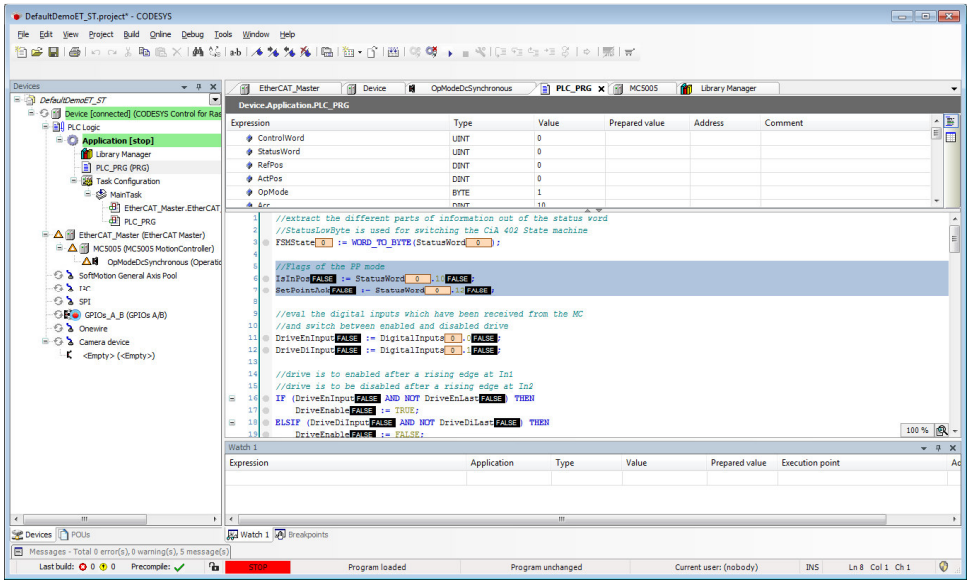


Figure 23 Debugging view of PLC_PRG - Application still stopped

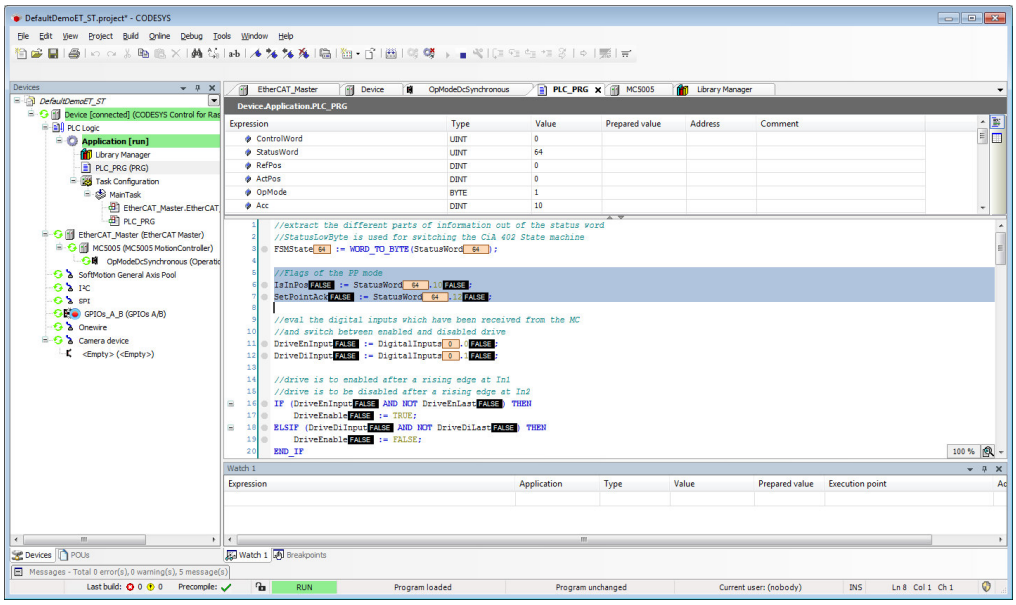


Figure 24 Debugging view of PLC_PRG - Application running

Startup of the EtherCAT subsystem will take a few moments. If the EtherCAT does not start up after a download/start a reset of the controller (Online/Reset warm) might help. Otherwise please check, whether the device in the project really is the one connected to the EtherCAT.

We can logout again using Online/Logout. The application can be explicitly stopped and restarted using the Debug menu.

Please note: The downloaded application will remain in the RasPi environment. It will be restarted automatically after the next reset of the RasPi.

Example Application

Overview

The purpose here is to start the drive in reaction to a digital input and cyclically move between two positions while also cyclically changing the profile parameters for acceleration and deceleration. The used OpMode is ProfilePosition Mode (0x6060.00 = 1).

This requires enabling the controller in a first step and then sending the target positions, waiting for being in position and updating the profile parameters. So there are several steps to be taken. A well suited solution pattern for such a problem is a step sequence. A step sequence is a pattern, where only a part of the PLC program is executed in each update cycle, depending of the step in which the program is. There is a special diagram to design these step sequences: sequential function chart (SFC). Here however we use ST and implement the step sequence using a

```
If (StepVariable = xxx) Then
…
ELSEIF (StepVariable = xxx) Then
…
END_IF
```

Implement the steps.

Each step may have actions to be taken only at the entry into a step, code to be executed while within a step and actions to be taken when switching to the next step. Obviously there should also be a condition when to switch to the next step. And there can be different conditions to branch into different next steps.

Examples on how to create a step sequence in a CODESYS environment using sequential function chart (SFC) can be found at www. youtube.com.

The main sequence of this application is shown in the flowchart of Figure 26. After preprocessing the inputs (control word and digital inputs) a first step sequence deals with the finite state machine of the servo drive according to CiA 402.

The control word is used to step the MotionController from the initial disabled state to fully operational state. The commands in the controlword depend on the statusword received from the drive unit.

The main idea is: If we have been in disabled state (stored in the variable DriveEnabled) and the enable switch at the discrete I/Os (DigIn1) of the MotionController is pressed, we send a sequence of enabling commands to the MotionControllers controlword. Steps are taken depending on the state signaled by the statusword.

On the other hand, if we are already enabled and receive a disable command from DigIn2, send a shutdown command to the controller to disable the operation once again.

Only after the state machine reached the operation enabled state, the DriveEnabled variable is set to true and the second step sequence for updating the position references is executed.

So finally – when the drive is enabled – the position references are sent to the drive. The first position reference of a sequence is always sent as an immediate value – which will be processed by the drive without waiting for any earlier commands to be finished. All subsequent references are sent only if the previous one has been reached.

Again there is a handshake between the PLC and the MotionController using bits in the controword and the statusword. The handshake between the PLC and the servo drive is shown in Figure 25. After a new reference position has been set in 0x607A.00 Target Position the New Setpoint bit in the controlword 0x6040.00 has to be set (in fact a rising edge is required to start the move). Only after this rising edge is received by the drive the new reference is acknowledged via the setpoint acknowledge bit in the statusword and the drive starts the move. It is Important to release the New Setpoint bit once again to create a next rising edge for the next position.

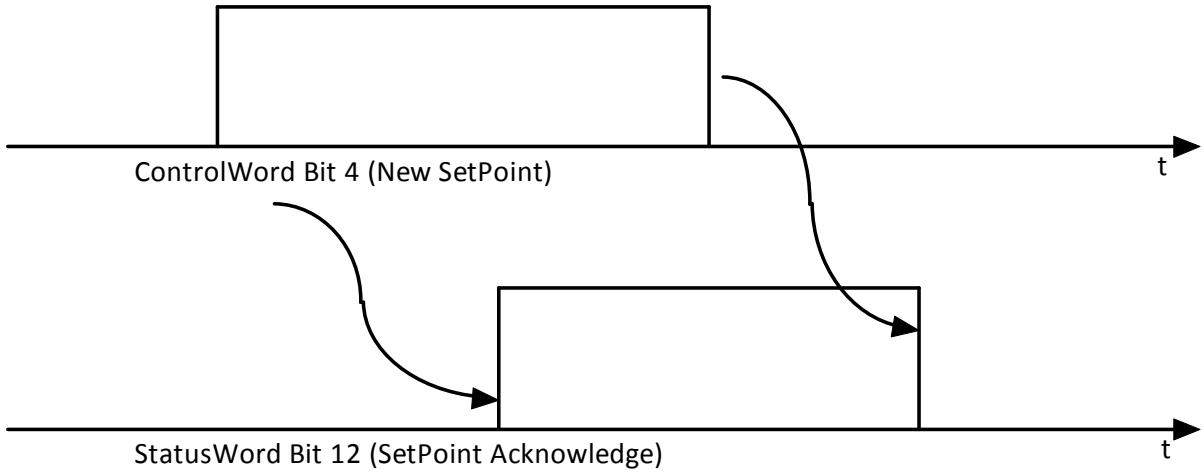


Figure 25 Handshaking sequence between the PLC (ControlWord) and the servo drive (StatusWord)

If the new setpoint is flagged as an immediate one (bit 5 of the controlword), the new command is immediately executed even if a previous command is still ongoing. This feature is used here for the very first command.

All subsequent commands are sent, only if the drive has reached the previous target position and signaled this by setting the Target Reached bit in the statusword.

So once again a step sequence is well suited to organize the different steps of this interaction.

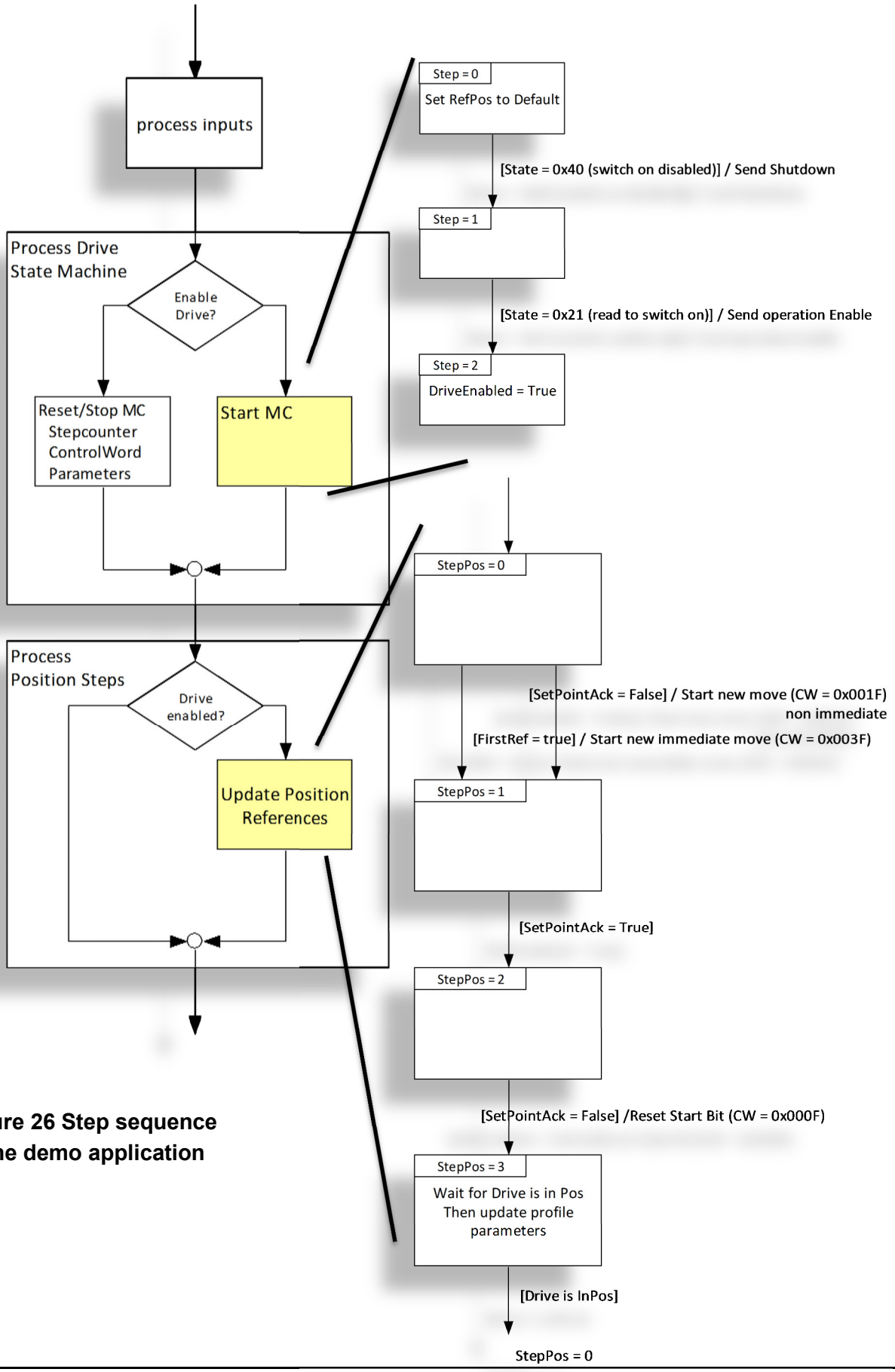


Figure 26 Step sequence of the demo application

Listing

Variables

This part is displayed in the upper part of the editor window and holds the definition of all variable instances.

```
VAR
      ControlWord: UINT:=0;
      StatusWord: UINT;
      RefPos : DINT:= 0;
      ActPos : DINT;
      OpMode : BYTE := 1;

      Acc: DINT:=10;
      Dec: DINT:=10;
      ProfileSpeed: DINT:= 5000;

      DriveEnable : BOOL:= FALSE;
      FSMState: BYTE;


      IsInPos: BOOL;
      SetPointAck: BOOL;
      DriveEnabled: BOOL:=FALSE;

      DigitalInputs : BYTE;
      DriveEnInput : BOOL;
      DriveEnLast : BOOL := FALSE;
      DriveDiInput : BOOL;
      DriveDiLast : BOOL := FALSE;

      FirstRef: BOOL:=TRUE;

      StepEn: INT:=0;
      StepPos: INT:=0;

      AccDecSlope: INT:=0;

END_VAR

VAR CONSTANT
      MaxAccDec:DINT:=500;
      MinAccDec:DINT:=50;
      DeltaAccDec:DINT:=10;
END_VAR
```

Code

The code is displayed in the lower part of the editor window and contains the functional logic of the application.

```
//extract the different parts of information out of the status word
//StatusLowByte is used for switching the CiA 402 State machine
FSMState := WORD_TO_BYTE(StatusWord);

//Flags of the PP mode
IsInPos := StatusWord.10;
SetPointAck := StatusWord.12;

//eval the digital inputs which have been received from the MC
//and switch between enabled and disabled drive
DriveEnInput := DigitalInputs.0;
DriveDiInput := DigitalInputs.1;

//drive is to enabled after a rising edge at In1
//drive is to be disabled after a rising edge at In2
IF (DriveEnInput AND NOT DriveEnLast) THEN
      DriveEnable := TRUE;
ELSIF (DriveDiInput AND NOT DriveDiLast) THEN
      DriveEnable := FALSE;
END_IF

//Store the flags to detect edges in the next step
DriveEnLast := DriveEnInput;
DriveDiLast := DriveDiInput;

//step through the CiA 402 drive state machine
//implemented as a step sequence station at StepEn = 0
IF (DriveEnable) THEN
      IF (StepEn = 0) THEN
            RefPos:=10000;
            IF (FSMState = 16#40) THEN
                  //if state is switch on disabled, send the switch on
                  ControlWord := 16#0006;
                  StepEn := 1;
            END_IF
      ELSIF (StepEn = 1) THEN
            IF (FSMState = 16#21) THEN
                  //if state is switched on, send enable operation
                  ControlWord := 16#000F;
                  StepEn := 2;
            END_IF
      ELSIF (StepEn = 2) THEN
            IF (FSMState = 16#27) THEN
                  //if   state   is   operation   enabled,   denote   the   drive
                  //being enabled
                  DriveEnabled := TRUE;
```

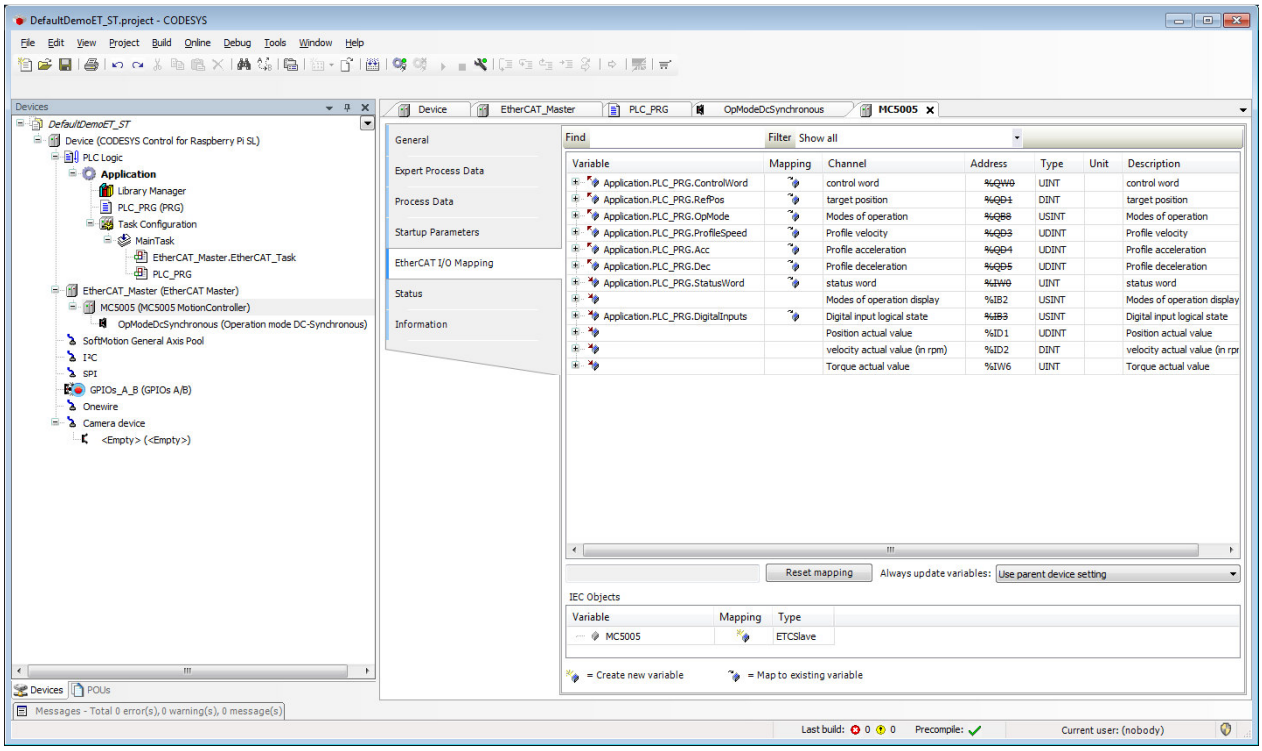
```
            END_IF
      END_IF
ELSE
      //drive shall be switched of or stay switched of
      //reset all variables to default values
      ControlWord := 0;
      StepEn := 0;
      DriveEnabled := FALSE;
      StepPos := 0;
      FirstRef := TRUE;
END_IF

//step sequence with the different position commands in PP mode
//first command is issued immediate
//the following commands are issed after having reached the previous target

//PP steps are sent only, if drive is enabled
IF (DriveEnabled) THEN
      //step 0: signal a new ref. This is either the default start value
      // set during enabling the drive
      //or the one calculated in StepPos step 3
      IF (StepPos = 0) THEN
            //if this is the first step after boot, this is sent flagged
            //as an immediate reference
            IF (FirstRef) THEN
                  ControlWord := 16#003F;
                  StepPos := 1;
                  FirstRef := FALSE;
            ELSE
                  //otherwise (not the first step) this a non immediate ref
                  IF (SetPointAck) THEN
                        //new ref has been acknowleged, so reset the
                        //new setpoint flag until the acknowledge is reset
                        ControlWord := 16#000F;
                  ELSE
                        //signal the new ref and step to the next step
                        // only, if the Ack Bit has not been set
                        ControlWord := 16#001F;
                        StepPos := 1;
                  END_IF
            END_IF
      ELSIF (StepPos = 1) THEN
            //last action in step 0 was, to signal the new ref, so we have
            // to wait, until the ref has been acknowlegded
            //waiting for acknowledge of pos
            //only then we can switch to the next step
            IF (SetPointAck) THEN
                  StepPos := 2;
            END_IF
      ELSIF (StepPos = 2) THEN
```

```
            //setpoint has been acknowlegeded, new setpoint bit can be
            // reset now
            //we should wait until the ack bit has been reset too
            ControlWord := 16#000F;
            IF (NOT SetPointAck) THEN
                  StepPos := 3;
            END_IF
      ELSIF (StepPos = 3) THEN
            //wait for InPos
            IF (IsInPos) THEN
                  //new pos has been reached
                  //now inivert the ref and change the profile parameters
                  // for the next step
                  RefPos :=(-RefPos);
                  //update profile Values
                  IF (AccDecSlope > 0) THEN
                        //we are still accelerating
                        IF (Acc < (MaxAccDec - DeltaAccDec)) THEN
                              Acc := Acc + DeltaAccDec;
                              Dec := Acc;
                        ELSE
                              AccDecSlope := 0;
                        END_IF
                  ELSE
                        //we slow down the dynamic
                        IF (Acc > (MinAccDec + DeltaAccDec)) THEN
                              Acc := Acc - DeltaAccDec;
                              Dec := Acc;
                        ELSE
                              AccDecSlope := 1;
                        END_IF
                  END_IF
                  //new profile values are calculated,
                  //we do now reset the sequence
                  StepPos := 0;
            END_IF
      END_IF
END_IF
```

Assignment of PLC variables to the process image



Variants

Add a homing sequence

The example above was about doing position control using PP mode. However there was no homing sequence used to reset the actual positon to a defined position.

Different types of homings are available for the FAULHABER MotionController. Here we might use homing on a lower limit switch: 0x6098.00 homing method = 17. As the homing method usually does not change for a given application the method can be set via the MotionManager during system configuration and saved as a part of the systems application parameters.

However we need to add some additional steps in our sequence:

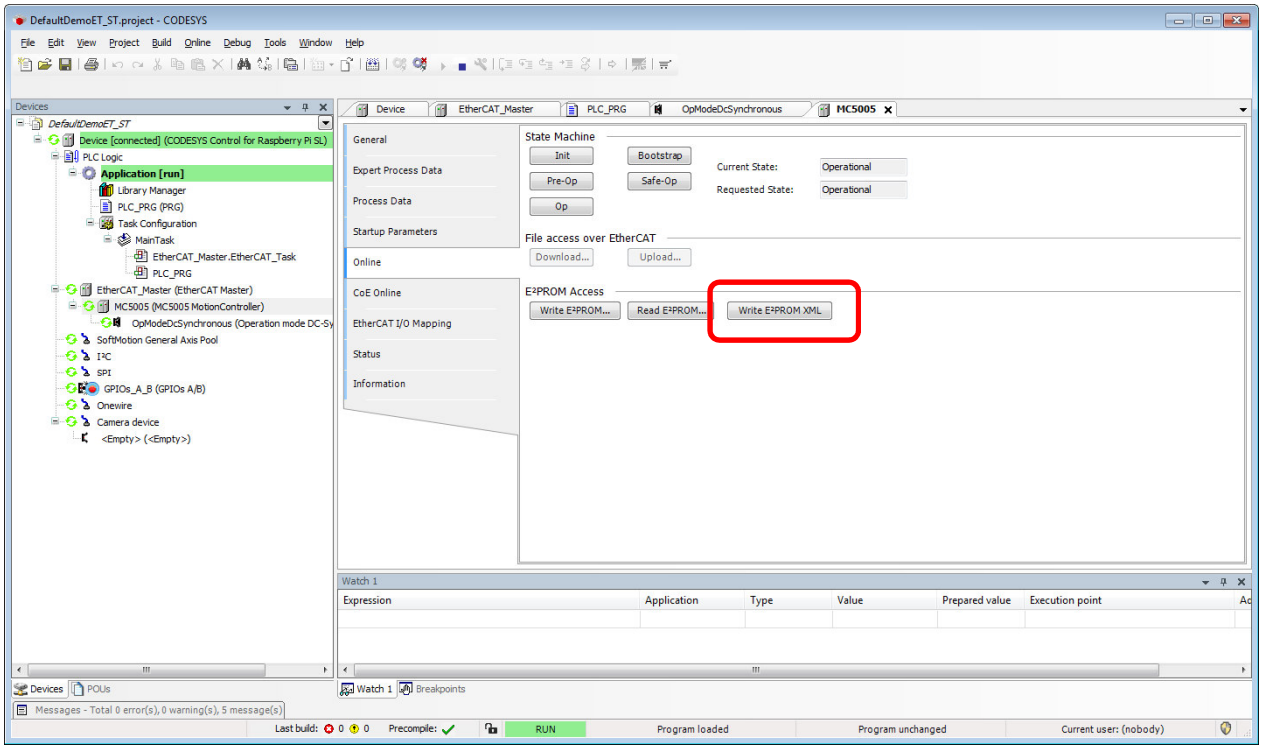
1. Switch OpMode to 0x6060.00 = 6: Homing
2. Start the homing sequence by setting the start bit in the controlword 0x6040.00
   0x6040.00 ← ControlWord = 0x00 1F
3. Wait for the homing being finished successfully by monitoring Bit 12 in the statusword 0x6041.00
   if(0x6041.00 → StatusWord.12 = true) Then …
4. Save the successfully executed homing in a local flag,
   switch OpMode to 0x6060.00 = 1 (PP) and
   start the UpdatePosition sequence according to Figure 26

As we already covered the OpMode in the mappings of Table 3 we don't even need to edit the PDO mappings in this step.

EEPROM Update

As mentioned the revision number stored in the EtherCAT slave (ESC) EEPROM should be identical to the revision of the firmware. Therefore after a FW update it is necessary to update the data stored in the ESC EEPROM too.

This can be done, when being connected to a running application or using the Online Config mode. In both cases the update is done using the Online tab of the MotionController settings. Simply press Write EEPROM XML will download the information out of the XML file used during system definition step.



Adding a Visualization

CODESYS allows easily adding a target visualization. Examples can be found at youtube. Visualization is added via the context menu off the Application (Figure 27). The visualization is then added as a separate component to the application tree (Figure 28). A separate task is created also in this step.

The controls for numeric values, switches or lamps can then be added and edited via drag and drop out of the visualization toolbox. Of course, you will need to connect them to variables of the application. This is a convenient way to add a user interface even if no external switches are available in a demo environment. The visualization can both be displayed and used in the CODESYS engineering environment or directly using a web browser:

The target address is <IP number of the target controller>:8080/webvisu.htm e.g.:

192.168.0.45:8080/webvisu.htm

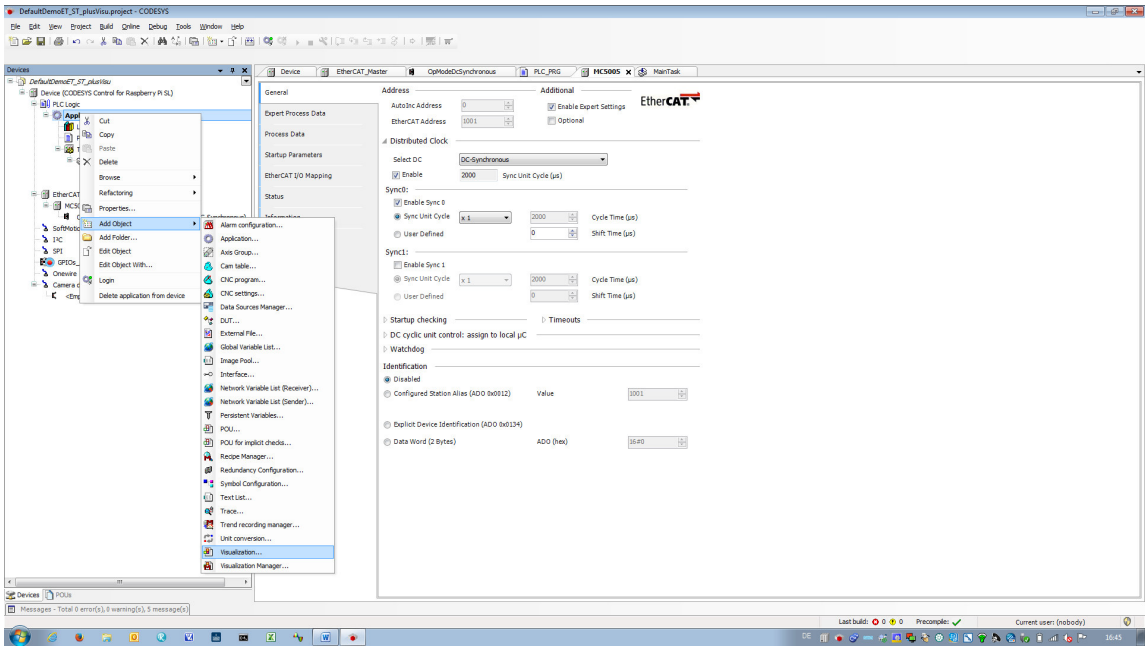


Figure 27 Add a visualization to an application

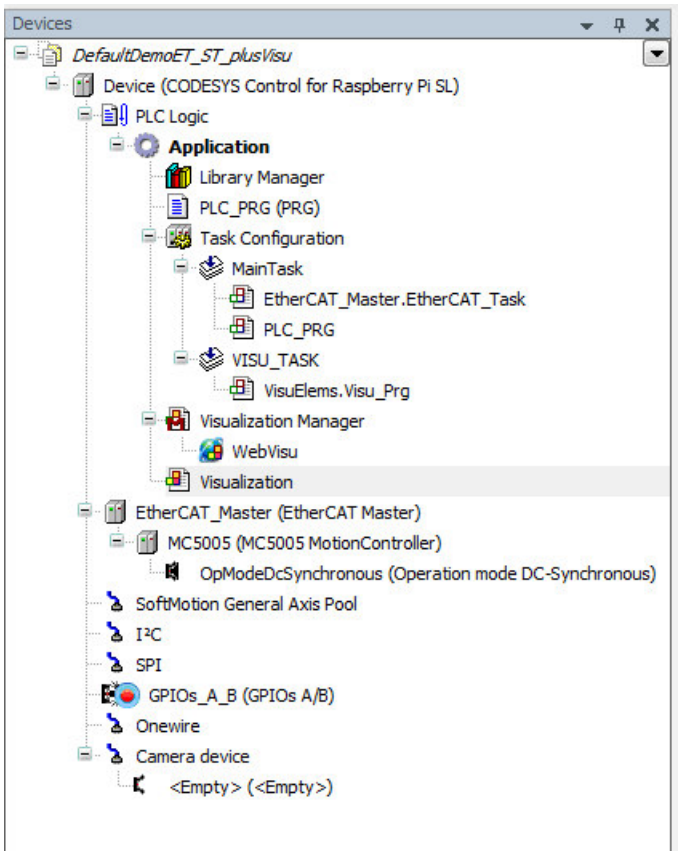


Figure 28 Application tree of a demo unit having target visualization

Additional Resources

FAULHABER Application Notes	
App-Note 150	How to control a FAULHABER MC V3.0 ET using TwinCAT
App Note 154	Updating the EtherCAT slave EEPROM using TwinCAT

FAULHABER manuals at www. faulhaber.com/manuals

FAULHABER demo systems at youtube

Rechtliche Hinweise

Urheberrechte. Alle Rechte vorbehalten. Ohne vorherige ausdrückliche schriftliche Genehmigung der Dr. Fritz Faulhaber & Co. KG darf insbesondere kein Teil dieser Application Note vervielfältigt, reproduziert, in einem Informationssystem gespeichert oder be- oder verarbeitet werden.

Gewerbliche Schutzrechte. Mit der Veröffentlichung der Application Note werden weder ausdrücklich noch konkludent Rechte an gewerblichen Schutzrechten, die mittelbar oder unmittelbar den beschriebenen Anwendungen und Funktionen der Application Note zugrunde liegen, übertragen noch Nutzungsrechte daran eingeräumt.

Kein Vertragsbestandteil; Unverbindlichkeit der Application Note. Die Application Note ist nicht Vertragsbestandteil von Verträgen, die die Dr. Fritz Faulhaber GmbH & Co. KG abschließt, soweit sich aus solchen Verträgen nicht etwas anderes ergibt. Die Application Note beschreibt unverbindlich ein mögliches Anwendungsbeispiel. Die Dr. Fritz Faulhaber GmbH & Co. KG übernimmt insbesondere keine Garantie dafür und steht insbesondere nicht dafür ein, dass die in der Application Note illustrierten Abläufe und Funktionen stets wie beschrieben aus- und durchgeführt werden können und dass die in der Application Note beschriebenen Abläufe und Funktionen in anderen Zusammenhängen und Umgebungen ohne zusätzliche Tests oder Modifikationen mit demselben Ergebnis umgesetzt werden können.

Keine Haftung. Die Dr. Fritz Faulhaber GmbH & Co. KG weist darauf hin, dass aufgrund der Unverbindlichkeit der Application Note keine Haftung für Schäden übernommen wird, die auf die Application Note zurückgehen.

Änderungen der Application Note. Änderungen der Application Note sind vorbehalten. Die jeweils aktuelle Version dieser Application Note erhalten Sie von Dr. Fritz Faulhaber GmbH & Co. KG unter der Telefonnummer +49 7031 638 688 oder per Mail von mcsupport@faulhaber.de.

Legal notices

Copyrights. All rights reserved. No part of this Application Note may be copied, reproduced, saved in an information system, altered or processed in any way without the express prior written consent of Dr. Fritz Faulhaber & Co. KG.

Industrial property rights. In publishing the Application Note Dr. Fritz Faulhaber & Co. KG does not expressly or implicitly grant any rights in industrial property rights on which the applications and functions of the Application Note described are directly or indirectly based nor does it transfer rights of use in such industrial property rights.

No part of contract; non-binding character of the Application Note. Unless otherwise stated the Application Note is not a constituent part of contracts concluded by Dr. Fritz Faulhaber & Co. KG. The Application Note is a non-binding description of a possible application. In particular Dr. Fritz Faulhaber & Co. KG does not guarantee and makes no representation that the processes and functions illustrated in the Application Note can always be executed and implemented as described and that they can be used in other contexts and environments with the same result without additional tests or modifications.

No liability. Owing to the non-binding character of the Application Note Dr. Fritz Faulhaber & Co. KG will not accept any liability for losses arising in connection with it.

Amendments to the Application Note. Dr. Fritz Faulhaber & Co. KG reserves the right to amend Application Notes. The current version of this Application Note may be obtained from Dr. Fritz Faulhaber & Co. KG by calling +49 7031 638 688 or sending an e-mail to mcsupport@faulhaber.de.