Instructions

How to set up a WEBER C30S with EtherCAT fieldbus on Beckhoff TwinCAT 3

1 ESI File for TwinCAT

Use the `ABC_CAN_ETHERCAT_V_x_yy.xml` file and copy it in the ESI (EtherCAT Slave Information) directory of TwinCAT 3. The path is normally located on Your local PC under `C:\TwinCAT\3.1\Config\Io\EtherCAT`.

The File is used by TwinCAT 3 to know about the C30S EtherCAT device.

2 Hardware connection

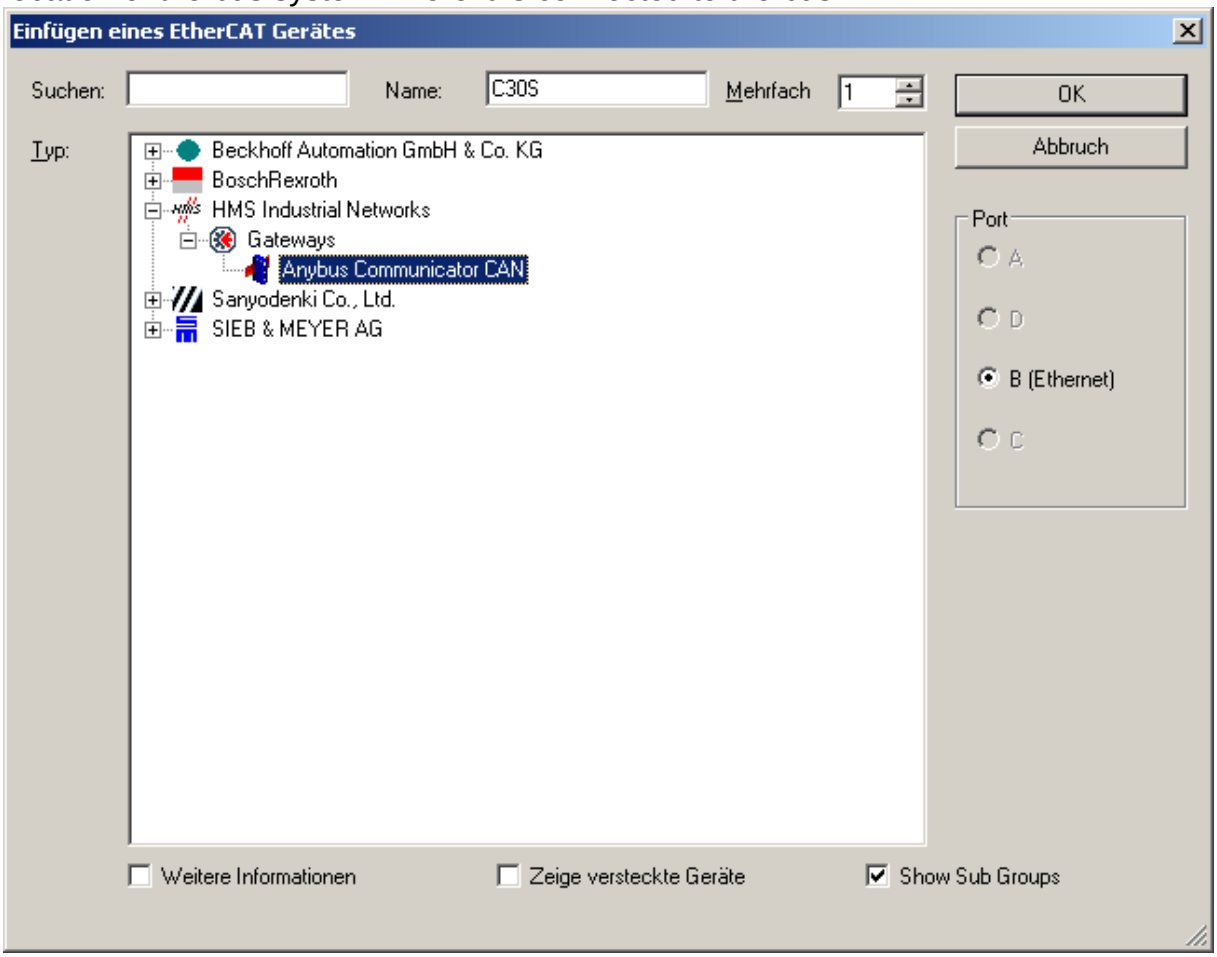
Make sure that the C30S is connected to the EtherCAT Network and the unit is switched on.

The lower Ethernet port (Port1) is the In while the upper port (Port2) is the Out. More information on the hardware connection can be found in the C30S manual.

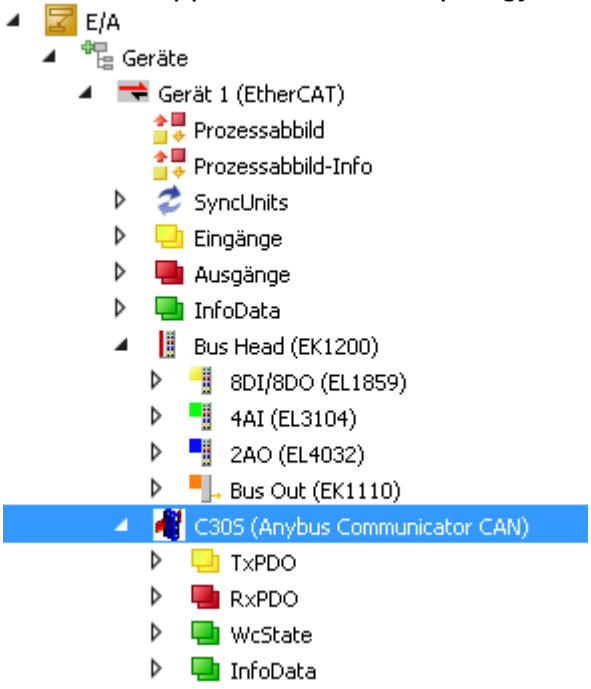
3 Configure a C30S slave

3.1 Insert the C30S bus coupler

In the TwinCAT 3 project at the E/A tree add the `Anybus Communicator CAN` at the location of the bus system where it is connected to the bus:



It will then appear at the bus topology:

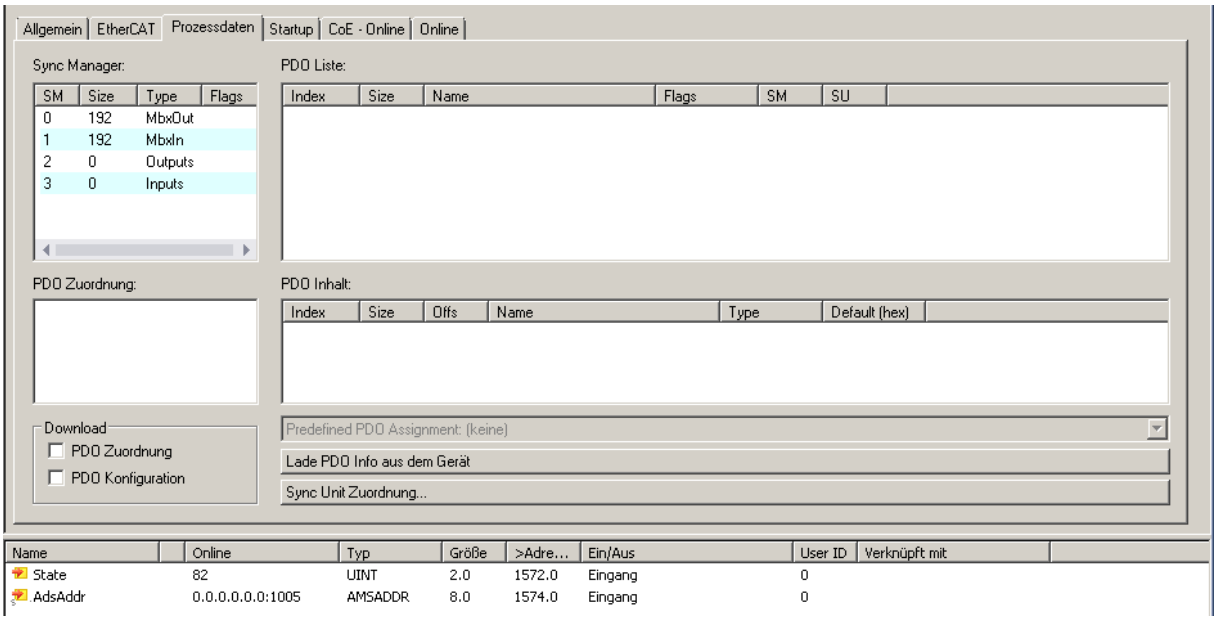


Name the Device C30Sxx, so You can identify it later.

3.2 Configure the I/O Data

The I/O data has a fixed definition in the C30S. To get those use the following procedure:

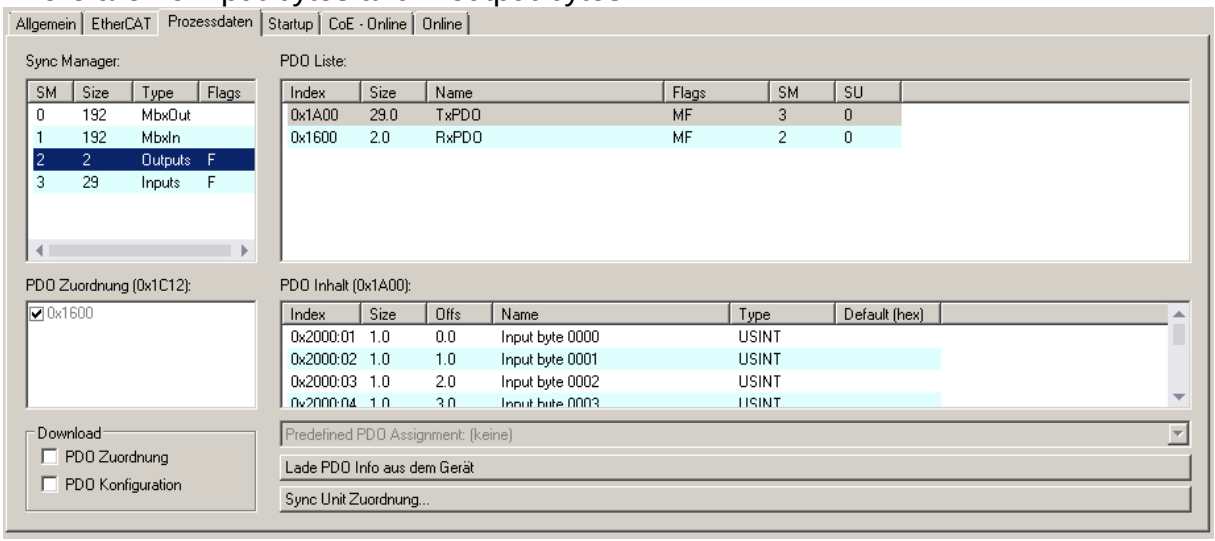
Please select the C30S participant on the bus, so You can view it's properties. Select the Tab `Process Data`.



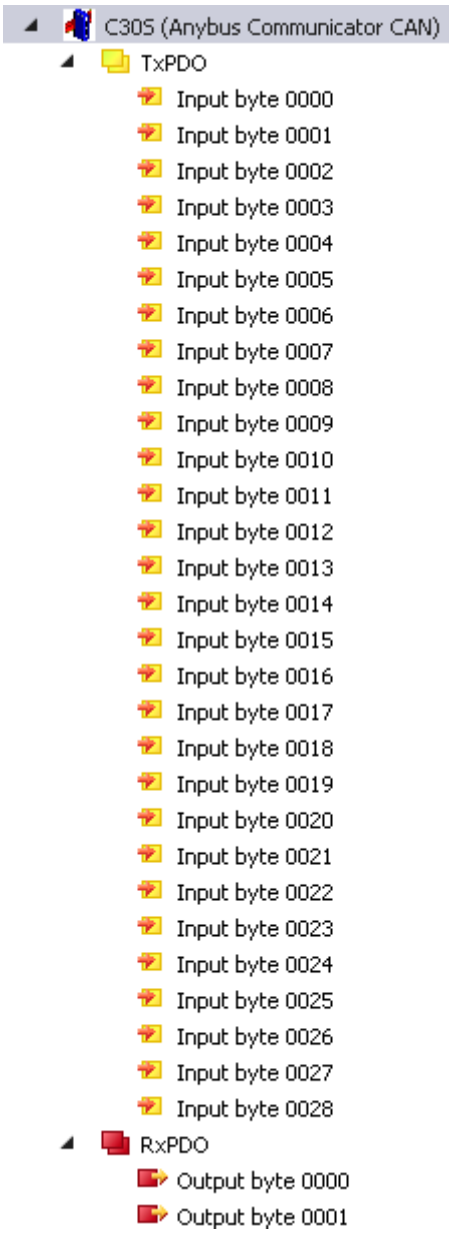
Now press the `Load PDO info from device` button.

TwinCAT will determine all the I/O bytes and inserts those.

There are 29 input bytes and 2 output bytes:



Now the I/O data should be showing up at the C30S hardwawe tree:



3.3 Use the data

As the date is now available as a byte array under the PLC Hardware tree, we need to sort out the values at the byte positions. The C30S manual is pointing out those in the Interface description.

3.3.1 Create data areas and make the mapping

An easy way to map the data is to create a byte array with the same size, as the C30S data in Your code:

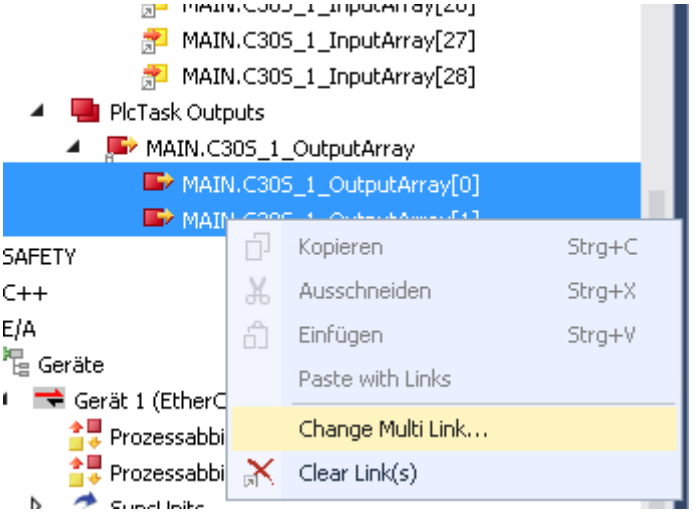
```
C30S_1_InputArray AT%I*: ARRAY[0..28] OF BYTE;
```

```
C30S_1_OutputArray AT%Q*: ARRAY[0..1] OF BYTE;
```

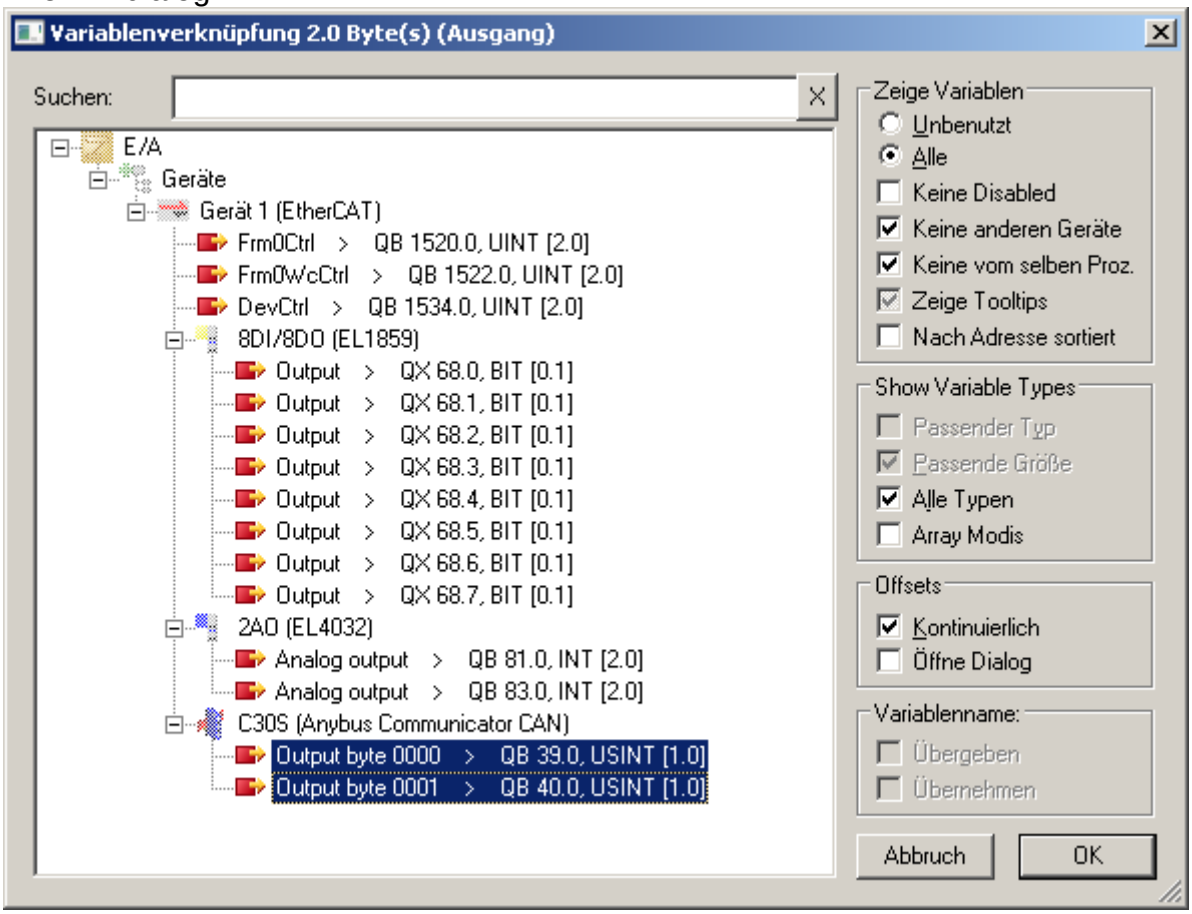
Now select all 28 input bytes and map them with the `Change Multi Link...` function to the 28 output bytes of the C30S hardware.

Same thing with the 2 output bytes, which need to be connected with the 2 input bytes of the C30S hardware.

Here the example for the 2 output bytes:



The link dialog:



Now all hardware bytes are represented by arrays in Your code. You can access all the bytes there.

3.3.2 Easy access to values

As we have only the byte arrays, we'd like to have a more practical way to access the C30S values. Especially on Intel platforms there is also the byte swapping, which needs to be done to interpret the values.

With the following structures the data can be accessed in a comfortable way. We define a structure for the C30S outputs, which become inputs to the PLC:

```
{attribute 'pack_mode' := '1'}
TYPE C30SOut_To_PLCIn :
STRUCT
	NoFault:BIT;
	ReadyToStart:BIT;
	OK:BIT;
	NOK:BIT;
	DepthReached:BIT;
	Reserve0:BIT;
	Reserve1:BIT;
	Reserve2:BIT;
	Result_Prg:BYTE;
	Result_DiagType:BYTE;
	Result_Code:UINT;
	Result_Torque:REAL;
	Result_PreTorque:REAL;
```

```
      Result_Angle:REAL;
      Result_Depth:REAL;
      Result_ScrewTime:REAL;
      Result_CycleNo:UDINT;
END_STRUCT
END_TYPE
```

As well we define a structure for the C30S inputs, which are outputs from the PLC:

```
TYPE C30SIn_To_PLCOut :
STRUCT
      Automatic:BIT;
      Start:BIT;
      AckcoledgeFault:BIT;
      TM1:BIT;
      TM2:BIT;
      ExternDigital:BIT;
      Reserve1:BIT;
      Reserve2:BIT;
      Prg:BYTE;
END_STRUCT
END_TYPE
```

If the byte swapping is necessary, the code could be looking like this in the Init section:

```
PROGRAM MAIN
VAR
      C30S_1_InputArray AT%I*: ARRAY[0..28] OF BYTE;
      C30S_1_OutputArray AT%Q*: ARRAY[0..1] OF BYTE;
      InputsFromC30s_1 :C30SOut_To_PLCIn;
      OutputsToC30s_1 :C30SIn_To_PLCOut;
      fb_Swap2Bytes: Swap2Bytes;
      fb_Swap4Bytes: Swap4Bytes;
END_VAR
```

The Code section:

```
//perform the byte swapping for all inputs, if runs on a intel processor
fb_Swap2Bytes( in_bytes:=ADR(C30S_1_InputArray[3]));
fb_Swap4Bytes( in_bytes:=ADR(C30S_1_InputArray[5]));
fb_Swap4Bytes( in_bytes:=ADR(C30S_1_InputArray[9]));
fb_Swap4Bytes( in_bytes:=ADR(C30S_1_InputArray[13]));
fb_Swap4Bytes( in_bytes:=ADR(C30S_1_InputArray[17]));
fb_Swap4Bytes( in_bytes:=ADR(C30S_1_InputArray[21]));
fb_Swap4Bytes( in_bytes:=ADR(C30S_1_InputArray[25]));
//copy over the data to the struct, to get easy access
MEMCPY(ADR(InputsFromC30s_1), ADR(C30S_1_InputArray), 29);

//insert here your step ladder code, to evaluate the C30S inputs and set
the outputs accordingly
OutputsToC30s_1.Start:=FALSE; //reset start singal
OutputsToC30s_1.Prg:=10; //set programm
```

```
//copy the values from the struct to the data array
MEMCPY(ADR(C30S_1_OutputArray), ADR(OutputsToC30s_1), 2);
```

If the PLC runs on a Motorola processor, the swapping has to be omitted!

3.3.3 Swap FB's

For the byte swapping the following function blocks can be used:

3.3.3.1 2 Byte swapping

Init Section:

```
FUNCTION_BLOCK Swap2Bytes
VAR_INPUT
      in_bytes: POINTER TO ARRAY[0..1] OF BYTE;
END_VAR
VAR_OUTPUT
END_VAR
VAR
      temp: BYTE;
END_VAR
```

Code section:

```
temp:=in_bytes^[0];

in_bytes^[0]:=in_bytes^[1];
in_bytes^[1]:=temp;
```

3.3.3.2 4 byte swapping

Init Section:

```
FUNCTION_BLOCK Swap4Bytes
VAR_INPUT
      in_bytes: pointer TO ARRAY[0..3] OF BYTE;
END_VAR
VAR_OUTPUT
END_VAR
VAR
      temp: ARRAY[0..1] OF BYTE;
END_VAR
```

Code section:

```
temp[0]:=in_bytes^[0];
temp[1]:=in_bytes^[1];

in_bytes^[0]:=in_bytes^[3];
in_bytes^[1]:=in_bytes^[2];
in_bytes^[2]:=temp[1];
in_bytes^[3]:=temp[0];
```