

Twincat Library: Accessing Drive Registers

Application Note

Version	Date	Editor	Comment
004	2015-10-13	dg	Update register names
005	2017-11-20	mvx	Add EtherCAT versions
006	2019-08-05	mvx	Note on committing parameters

Document AN109_TwinCAT-AccessingDriveRegisters_EP
 Version 006
 Source Q:\doc\ApplicationNotes\
 Destination T:\doc\ApplicationNotes
 Owner mvx

Copyright © 2017	Triamec Motion AG	Phone +41 41 747 4040
Triamec Motion AG	Lindenstrasse 16	Email info@triamec.com
All rights reserved.	6340 Baar / Switzerland	Web www.triamec.com

Disclaimer

This document is delivered subject to the following conditions and restrictions:

- This document contains proprietary information belonging to Triamec Motion AG. Such information is supplied solely for the purpose of assisting users of Triamec products.
- The text and graphics included in this manual are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.
- Information in this document is subject to change without notice.

Table of Contents

1 Target and Purpose.....1	3.1 Find the COE address.....3
2 PLC Code for Trialink devices.....2	3.2 Using the Triamec library.....3
3 EtherCAT devices.....3	3.3 Using standard TwinCAT functions.....4
	4 Committing Parameters.....5

1 Target and Purpose

The Triamec TwinCat library comes with basic sample codes for NCI and CNC. This application note describes additional functions available in this library.

This note describes how to read and write to drive (dsp or fpga) registers through an edge trigger. If an

application requires cyclic information at a high rate, consider AN105.

2 PLC Code for TrialLink devices

The code below sets the motor **temperature sensor type** and **-limit** and reads the **temperature**.

The SET input of **TL_MC_RegisterWrite** depends on the input data type. Use

- setBin to specify an integer value
- setFloat to specify a float value
- The "setBin" version must be used, whenever $0 <> \text{setReg AND TL_REG_FLOAT32_MASK}$

The GET outputs of **TL_MC_RegisterRead** depend on the data type. Use

- GetBin to read an integer value (DWORD)
- GetFloat to read a float or integer value

The value is set or read after a positive edge on Execute.

Declaration in MAIN_SLOW

```
motorTemperature          : TL_MC_RegisterRead;
motorTemperatureLimit     : TL_MC_RegisterWrite;
motorSensorType           : TL_MC_RegisterWrite;
```

and the code

```
motorSensorType.Execute   := gAxis[4].ready;
motorSensorType.SetReg    := gAxis[4].MC_Axis.register.GenPar.MotorSensor.SensorType;
motorSensorType.SetBin    := TL_C.GenPar.MotorSensorType.KTY84;
motorSensorType(axis:=gAxis[4].MC_axis, Trialink:=Trialink);

motorTemperatureLimit.Execute := motorSensorType.Done AND (Trialink.Trialink.OccationalTimer <> 333);
motorTemperatureLimit.SetReg  := gAxis[4].MC_Axis.register.GenPar.MotorSensor.UpperErrorLevel;
motorTemperatureLimit.SetFloat := 80;
motorTemperatureLimit(axis:=gAxis[4].MC_axis, Trialink:=Trialink);

motorTemperature.Execute   := Communication_Ready;
motorTemperature.SetReg    := gAxis[4].MC_Axis.register.GenSig.MotorTemperature;
motorTemperature(axis:=gAxis[4].MC_axis, Trialink:=Trialink);
```

Note, that the IN_OUT parameter **axis** uses the PLCopen-Axis "**.MC_axis**" of type **TL_MC_AXIS_REF** not the general axis module "**gAxis[4]**" of type **TL_AxisSlow**.

Use intelliSense to find the possible registers and constants.

For TSD80 drives, the two registers are:

```
gAxis[4].MC_Axis.register2.Axis.Parameters.Motor.TemperatureSensorType
gAxis[4].MC_Axis.register2.Axis.Parameters.Motor.TemperatureUpperLimit
```


code to write an integer register

```
state0:
  write(axis:=axis, execute := FALSE);
State1:
  write.value      := 137;
  write.address := 16#238701;
  write(axis:=axis, execute := TRUE);
  IF write.done THEN
    write(axis:=axis, execute := FALSE);
  END_IF
```

3.3 Using standard TwinCAT functions

Add the library Tc2_EtherCAT and declare functions for read and write

```
netId      : STRING(23) := '192.168.10.99.2.1'; // see adapter netId
driveId     : UINT := 1001;           // see drive EtherCAT Addr
valueRd     : REAL;
valueWr     : LREAL := 1.2345678;
readReg     : FB_EcCoeSdoRead;
writeReg    : FB_EcCoeSdoWrite;
```

to read Application.Variables.Floats[1]

```
readReg.sNetId := netId;
readReg.nSlaveAddr := driveId;
readReg( nIndex:=16#2386, nSubIndex :=2, pDstBuf:= ADR(valueRd), cbBufLen:=SIZEOF(valueRd));
```

to write Application.Variables.Doubles[1]

```
writeReg.sNetId := netId;
writeReg.nSlaveAddr := driveId;
writeReg(nIndex:=16#2385, nSubIndex :=3, pSrcBuf:= ADR(valueWr),cbBufLen:=SIZEOF(valueWr));
```

Then set bExecute to FALSE and TRUE to start the transaction.

Please note that the type of the variables readReg and writeReg must correspond to the type of the Drive register (DINT, REAL, LREAL).

4 Committing Parameters

Changing one of these registers takes immediate action:

- Axes[].Commands.*
- Axes[].Information.*
- Axes[].Parameters.Motor.TemperatureSensorType and TemperatureUpperLimit

Other registers of type "Parameters", however, require an action to become active. This allows changing a set of controller parameters in a controlled manner. This commit action is commanded by setting the following Boolean to True.

- | | |
|--|-----------------------------------|
| ▪ Set | After changing parameters |
| ▪ General.Commands.CommitParameter | General |
| ▪ Axes[].Commands.PathPlanner.CommitParameter | PathPlanner |
| ▪ Axes[].Commands.PositionController.CommitParameter | PositionController or Commutation |
| ▪ Axes[].Commands.CurrentController.CommitParameter | CurrentController or Motor |

The Boolean is reverted to False automatically, when the commit finished. Do not change any parameters before the last commit finished.