
EpicsComModule
EPICS to TwinCAT 3 communication interface

SUMMARY

This manual describes a communication method between EPICS and Beckhoff TwinCAT 3 PLC. The communication is based on Ethernet TCP-IP where the EPICS side is configured using Stream Device and ASYN driver and the TwinCAT side is configured with a PLC module called EpicsComModule that utilizes the Beckhoff TCP-IP framework. Internal communication between different TwinCAT 3 modules is based on ADS-communication.

Variables in the TwinCAT PLC are accessed, both for reading and writing, through their symbolic names. In order to read a variable the command string should be formatted "*symbolic name?*;" and to write a variable "*symbolic name=value;*". If needed variables also can be accessed based on absolute addressing. This could for example be useful when reading or writing data to the Motion part of TwinCAT.

Data in different TwinCAT modules can be accessed by adding optional ADS-port information to the command string.

This manual describes how to configure the EpicsComModule, in a new or existing TwinCAT 3 project, and also partly the corresponding EPICS- configuration.

For more examples and information, please see the sample project for TwinCAT 3 and EPICS.

Note: A quick start guide is available in Chapter 13

Document Number 1
Date 2015-06-25

Date	Version	Author	Description
9/1-15	1	Anders Sandström	First version
2/3-15	2	Anders Sandström	Added chapter “Trouble shooting”, updated doc for use with EpicsComModule v0_20
24/4-15	3	Anders Sandström	Updated manual to fit version 0.9.0 of EpicsComModule

LIST OF ABBREVIATIONS

Abbreviation	Definition
EPICS	Experimental Physics and Industrial Control System
SCADA	Supervisory Control And Data Acquisition
PLC	Programmable Logic Controller
TCP-IP	Transmission Control Protocol-Internet Protocol
ADS	Automation Device Specification

TABLE OF CONTENT	PAGE
SUMMARY	2
LIST OF ABBREVIATIONS	4
1. INTRODUCTION	8
2. PROTOCOL.....	10
2.1. General.....	10
2.2. Writing	10
2.3. Reading	11
2.4. Options.....	12
2.4.1. ADSPORT	12
2.4.2. INTERRUPT	12
2.5. Static commands.....	13
2.5.1. ".THIS."-Command	13
2.5.2. ".ADR."-Command	15
2.5.2.1. Reading and writing.....	15
2.5.2.2. Reading absolute address of symbolic value	17
2.6. Supported native data types	18
2.7. Supported structures	19
3. TWINCAT 3 CONFIGURATION.....	20
3.1. General.....	20
3.2. Access to source and manuals	21
3.2.1. TwinCAT source Git (MCAG_Base_Project).....	21
3.2.2. Motor Record driver for TwinCAT source Git.....	21
3.3. TwinCAT MCAG_Base_Project configuration.....	21
3.3.1. Open TwinCAT solution.....	21
3.3.2. Installation of Beckhoff TCP/IP framework	21
3.3.3. TCP/IP framework Licencing	22
3.3.4. Build and download	22
3.3.5. Test communication.....	23
3.4. EpicsComModule Settings.....	24
4. EPICS CONFIGURATION	26

4.1.	proto-file	27
4.2.	db-file	28
4.3.	st.cmd-file	29
4.4.	Supported record types	30
4.5.	Motor Record Support	30
4.5.1.	EPICS.....	30
4.5.2.	TwinCAT	30
5.	PERFORMANCE	32
6.	RESTRICTIONS	33
6.1.	Communication buffer size	33
7.	EPICS CONFIGURATION FILE GENERATION TOOL.....	34
8.	FUTURE WORK/IDEAS	35
8.1.	Interrupt based communication	35
8.2.	Lookup table for addresses	35
8.3.	Binary communication	35
8.4.	Source code on Github.....	35
8.5.	Optimization of memory usage and performance	35
9.	TIPS AND TRICKS.....	36
9.1.	Telnet access	36
9.2.	Customization of the protocol	36
9.2.1.	General customization settings.....	36
9.2.2.	Settings for individual connection.....	37
9.2.2.1.	Return of variable name.....	37
9.2.2.2.	Return data on write	37
9.2.2.3.	Change of default ADS-Port.....	38
9.2.2.4.	Read Connection Index.....	38
9.3.	Change of default ADS-Port address	38
9.4.	Identify to which motion axis the FB_DriveVirtual is connected	39
9.4.1.	Identify ADS-Ports for PLC-Modules	41
10.	TROUBLE SHOOTING	42
10.1.1.	ADS error 1797	42
10.1.2.	Will not accept connections	42

10.1.3. Library mismatch	43
10.1.4. Performance issues	44
10.1.5. Create and link to new Task	45
11. REFERENCES	49
12. APPENDIX 1: SUPPORTED STRUCTURES	50
12.1. DUT_AxisStatus	50
12.2. DUT_HardwareStatus.....	51
12.3. DUT_ChopperAxisStatus	51
12.4. DCTIMESTRUCT	51
12.5. DUT_Stats.....	51
12.6. DUT_HardwareStatus_32Bits_v0_01	51
12.7. DUT_HardwareStatus_32Float_v0_01	51
12.8. DUT_ResolverCalib.....	51
13. APPENDIX 2: QUICK START GUIDE.....	52
13.1. TwinCAT	52
13.2. EPICS.....	52

1. INTRODUCTION

This document describes one way to communicate between an EPICS SCADA system, [1], and variables within a Beckhoff TwinCAT 3 controller. An overview of a typical setup is shown in Figure 1.

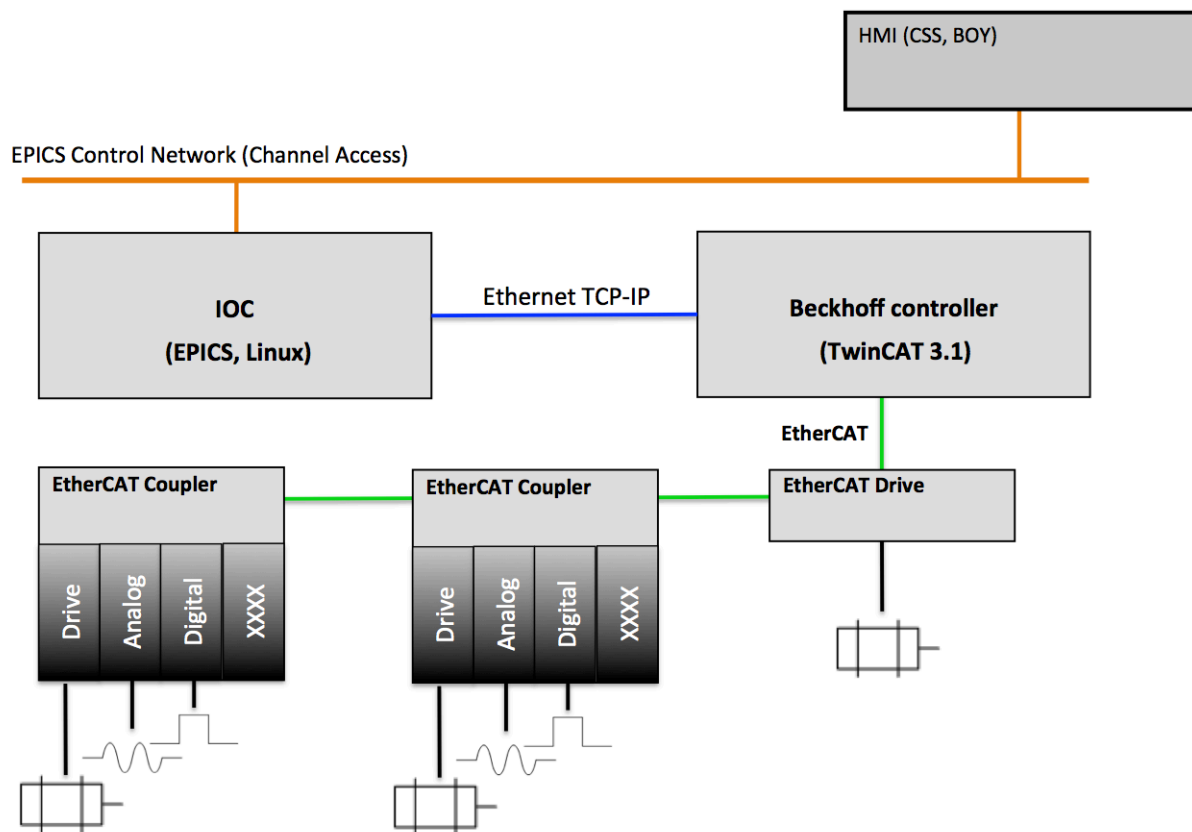


Figure 1: Typical communication setup between EPICS and TwinCAT 3

The communication is based on a TCP/IP connection between the EPICS IOC (TCP/IP-client) and TwinCAT 3 PLC (TCP/IP-server).

The EPICS side uses Stream Device and ASYN-driver to output and receive data in string format, see chapter 4 “EPICS Configuration”

In the TwinCAT 3 side, the module “EpicsComModule” needs to be loaded, chapter 3.

After loading the EpicsComModule native variables, and arrays of native variables, in the TwinCAT 3 PLC controller can be accessed and written from EPICS records. The EpicsComModule interface also supports a few different kinds of custom structures described in chapter 12.

Note: A quick start guide is available in Chapter 13.

Variables in the TwinCAT PLC are accessed in EPICS through their symbolic variable names (in the PLC). For reading the corresponding value is returned and for writing the status of the write command is returned to EPICS, see Figure 2.

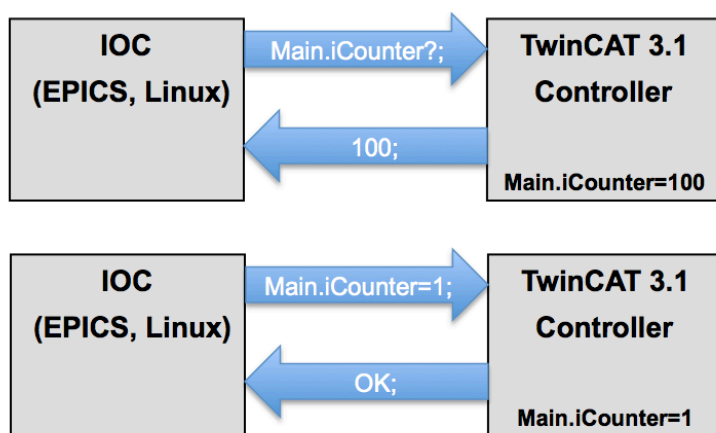


Figure 2: Communication concept for reading and writing

2. PROTOCOL

2.1. General

The communication is based on a TCP/IP connection using text strings.

The Stream Device text string must be formatted with certain syntax:

- Frame terminator = `\n` (linefeed = ASCII 10)
- Command separator = `“;”`
- Array element separator = `“,”`
- An option may be supplied before each command separated with `“/”`.

Example:

`“Option1/Command1;Command2;Option3/Command3;
”`

2.2. Writing

The EPICS command string for writing a value to the PLC should start with the symbolic name of the variable, an equal sign and ending with the value to be set.

The EpicsComModule in the PLC will return a string containing `“OK;”` if the write operation succeeded or an error code if the write operation failed:

Example 1:

EPICS Command string: `“Main.M1.fPosition=100; Main.M1.fVelocity=1000;”`

EpicsComModule will return: `“OK;OK;”`

Example 2 (array):

EPICS Command string: `“Main.fTestArray=1,2,3,4,5,6,7,8,9,10;”`

EpicsComModule will return: `“OK;”`

Note: The current implementation is restricted to write arrays with a maximum length of 20 Kbytes. This is a setting in the EpicsComModule (Global_Variables-> PLCPRJ_BUFFER_SIZE) and can be changed. For systems with TC performance level 50 or lower this setting can be reduced in order to reduce memory usage and CPU load.

2.3. Reading

The EPICS command string for reading a value from the PLC should start with the symbolic name of the variable and ending with a question mark:

The EpicsComModule in the PLC will return a string containing the value of the variable or an error message.

Example 1:

EPICS Command string: *"Main.M1.fPosition?; Main.M1.fVelocity?;"*

EpicsComModule will return: *"100;1000;"*

Example 2 (array):

EPICS Command string: *"Main.fTestArray?;"*

EpicsComModule will return: *"1,2,3,4,5,6,7,8,9,10;"*

2.4. Options

2.4.1. ADSPORT

Different modules within the TwinCAT controller are normally allocated different ADS-ports. The default ADS-port for internal communication is set to 851, which normally is the address of the first PLC-module in a TwinCAT 3 controller. If a different module needs to be accessed the “ADSPORT” option can be used.

The option defines the ADS-port only for the current command string. However, the default ADS-port of the EpicsComModule will not be changed.

Example of accessing two different modules in TwinCAT 3:

```
“ADSPORT=851/Main.M1.fPosition=100;ADSPORT=852/Main.iCounter?;
```

```
”
```

2.4.2. INTERRUPT

The interrupt option is currently not supported in the newest version of the EpicsComModule. A new implementation of event based data communication is in development. In this implementation the host can subscribe to a certain variable and will get an update if the variable changes.

2.5. Static commands

EpicsComModule supports some special commands:

1. ".THIS."
2. ".ADR."

2.5.1. ".THIS."-Command

The ".THIS." command allows access to data in the function block that handles the current communication connection (in the EpicsComModule PLC). Since the different connections are handled by an array of functions blocks the client normally don't know to what index in the array it is connected. The ".THIS." part of the command will be exchanged to "Main.fbServer.aApplications[X]." by the parser, where **X** is the array index of the current connection.

NOTE: The ".THIS."-command normally needs the ADSPORT option since the default ADS-Port number normally is not linked to the EpicsComModule port.

A typical command will look like:

```
"ADSPORT=852/.THIS.stSettings.nADSPort?;"
```

After parsing the command will look like (connection index=1):

```
"ADSPORT=852/Main.fbServer.aApplications[1].stSettings.nADSPort?;"
```

The command will then pass the normal parsing sequence for symbolic data access.

Some useful settings that can be accessed with ".THIS." are listed in Table 1.

Table 1: ".THIS." command useful variables

	Variable	Type	Read/Write	Default	Comment
1	stSettings.nADSPort	INT	R/W	851	Default ADS-Port for the current connection
2	stSettings.nConnectionIndex	UDINT	R		Index of current connection (in array of FB_ServerApplication)
3	stSettings.bReturnData	BOOL	R/W	FALSE	Return data on write commands (data is first written then read and sent to host)
4	stSettings.bReturnVarName	BOOL	R/W	FALSE	Return variable name when data is returned to host (always applied when reading structures)
5	sVersion	STRING	R/(W)	"0.9.0"	Version of FB_ServerApplication
6	sFeatures	STRING	R/(W)	"stv1"	A comma separated string describing special features (stv1=support for stAxisStructV1)
7	sError	STRING	R		Error string for command parser
8	bError	BOOL	R		Error bit for command parser
9	nErrorId	INT	R		Error Id number for command parser
10	nErrorCounter	INT	R		Error counter for command parser (since boot of PLC)

2.5.2. “.ADR.”-Command

The “.ADR.” command allows access to data by specifying the absolute address instead of a symbolic address. The command can also be used to find out the absolute address of a symbolic variable. This command is needed since not all data in a TwinCAT system is accessible by symbolic names. Typically changes of Motion Axis Parameters can easily be achieved with the ADR command.

2.5.2.1. Reading and writing

The “.ADR.” command has the following syntax for reading and writing:

“.ADR.IndexGrp,IndexOffset,Size,DataType?;”

“.ADR.IndexGrp,IndexOffset,Size,DataType=value;”

The “IndexGroup” and the “IndexOffset” parameters are hex values that when combined defines the absolute address of the data to access. These parameters can normally be accessed in the engineering tool by hovering over a parameter, see Figure 3, or by reading the TwinCAT manuals [2]. The “Size” parameter is the size of the data to be accessed in bytes.

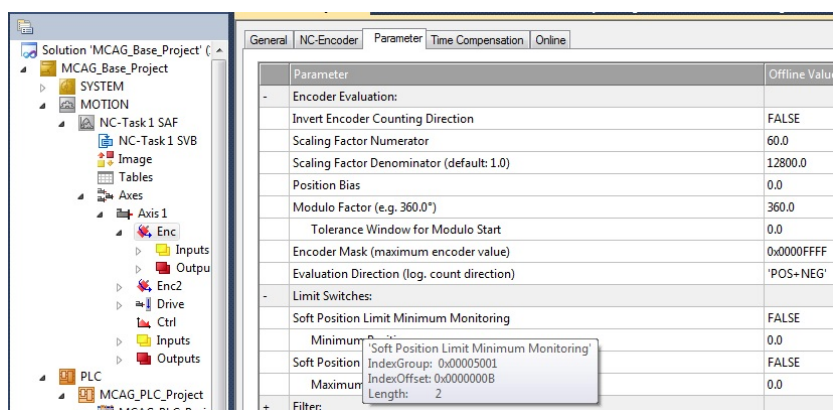


Figure 3: Index group and Index Offset

The “DataType” parameter is an index that defines the data type of the data addressed. Table 2 lists all available alternatives for the “DataType” index.

Table 2: Data type index for ADR command

	ADS Type	DataType	Description
1	ADST_VOID	0	VOID
2	ADST_INT8	16	SINT
3	ADST_UINT8	17	BYTE
4	ADST_INT16	2	INT

5	ADST_UINT16	18	UINT
6	ADST_INT32	3	DINT
7	ADST_UINT32	19	UDINT
8	ADST_INT64	20	LONG
9	ADST_UINT64	21	ULONG
10	ADST_REAL32	4	FLOAT
11	ADST_REAL64	5	DOUBLE
12	ADST_BIGTYPE	65	
13	ADST_STRING	30	STRING
14	ADST_WSTRING	31	WSTRING
15	ADST_REAL80	32	80 bit REAL
16	ADST_BIT	33	BOOL
17	ADST_MAXTYPES		

Some useful “.ADR.” commands are listed in Table 3.

Table 3: Example of useful ADR commands (ADSPORT=501 for the motion module)

	Command string	Description (n=1 for first axis)
1	“ADSPORT=501/.ADR.16#500n,16#B,2,2=1;”	Set enable of soft position limit minimum monitoring for axis n
2	“ADSPORT=501/.ADR.16#500n,16#C,2,2=1;”	Set enable of soft position limit maximum monitoring for axis n
3	“ADSPORT=501/.ADR.16#500n,16#D,8,5=10.0;”	Set soft limit minimum position to 10 for axis n
4	“ADSPORT=501/.ADR.16#500n,16#E,8,5=100.0;”	Set soft limit maximum position to 100 for axis n
5	“ADSPORT=501/.ADR.16#400n,16#27,8,5=100.0;”	Set maximum speed for axis n

2.5.2.2. Reading absolute address of symbolic value

For reading the address of a symbolic variable the following syntax should be used:

`".ADR.symbAdress?;"`

The result will be formatted like below:

`"IndexGrp,IndexOffset,Size,DataType;"`

Example 1:

EPICS Command string: `".ADR.Main.M1.bEnable?;"`

EpicsComModule will return: `"16#4040,16#7DE01,1,33;"`

NOTE: Absolute addresses will/can change each time you compile your PLC-software. Therefore caution has to be taken so that not the wrong address is accessed. However, normally accessing an absolute address can be faster than accessing a symbolic address.

2.6. Supported native data types

The supported native data types are listed in Table 4.

Table 4: Supported data types

	TwinCAT 3 data type	Size [bits]	Array
1	ADST_BIT	1	Yes
2	ADST_SINT8	8	Yes
3	ADST_UINT8	8	Yes
4	ADST_INT16	16	Yes
5	ADST_UINT16	16	Yes
6	ADST_INT32	32	Yes
7	ADST_UINT32	32	Yes
8	ADST_INT64	64	Yes
9	ADST_UINT64	64	Yes
10	ADST_REAL32	32	Yes
11	ADST_REAL64	64	Yes

* Strings are supported but arrays of strings (matrices) are not supported

2.7. Supported structures

The EpicsComModule supports reading of a few types of structures with a single command. The communication will be more efficient using structures compared to polling each value within a structure since all data will be returned within the same frame.

A tool has been developed, “TwinCATEPICSUtilities”, that facilitates efficient data transfer by automatic generation of configuration files for EPICS based on a TwinCAT structure or function block files.

Note: Currently only reading of structures is supported. Arrays of structures are not supported.

Premade EPICS templates files for using the listed structures are supplied with the EpicsComModule package.

For a list of supported structures see chapter 12.

3. TWINCAT 3 CONFIGURATION

3.1. General

The TwinCAT 3 PLC project module, EpicsComModule, handles the communication between EPICS and the TwinCAT 3 controller. The communication link to EPICS is achieved with a TCP-IP link and internal communication, between the different PLC-modules, is handled by ADS-communication, see Figure 2.

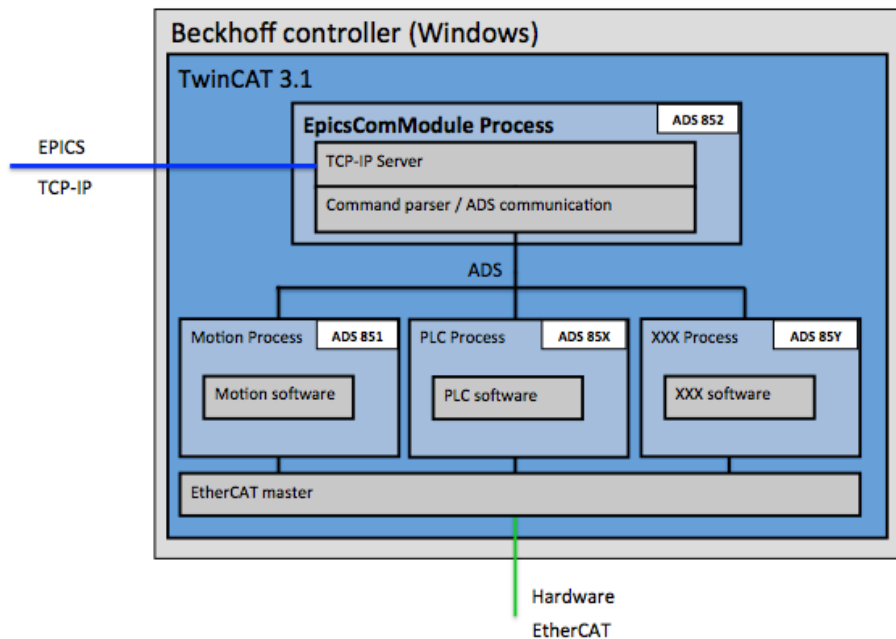


Figure 4: EpicsComModule internal and external communication.

The EpicsComModule uses the Beckhoff TCP-IP server framework for the external communication and ADS function blocks for internal ADS communication.

The EpicsComModule needs to be included in the TwinCAT 3 project to get a working communication link between EPICS and TwinCAT 3.

3.2. Access to source and manuals

The source code and manuals can be accessed on Github [5,6]:

3.2.1. TwinCAT source Git (MCAG_Base_Project)

Link to website: https://github.com/EuropeanSpallationSource/MCAG_Base_Project

“git clone https://github.com/EuropeanSpallationSource/MCAG_Base_Project.git”

3.2.2. Motor Record driver for TwinCAT source Git

Link to website: https://github.com/EuropeanSpallationSource/ESS_MCAGmotor

“git clone https://github.com/EuropeanSpallationSource/ESS_MCAGmotor.git”

A quick start manual is available in Chapter 13.

3.3. TwinCAT MCAG_Base_Project configuration

The source code and projects accessible on Github [5,6] are preconfigured to perform well on a mid range performance TwinCAT controller. However, sometimes it can be necessary to change some settings in order to optimize performance.

3.3.1. Open TwinCAT solution

The TwinCAT source is opened by double clicking the solution file, “MCAG_Base_Project.sln”, located in the top folder of the MCAG_Base_Project. The project will then be opened in the TwinCAT IDE (based on Visual Studio).

3.3.2. Installation of Beckhoff TCP/IP framework

The Beckhoff TCP/IP service application needs to be installed on the controller. This software can be downloaded from the Beckhoff website [4]. The installation procedure can be different depending on the OS running on the intended controller (Embedded Standard or Embedded compact). The way to install this software is described in the Beckhoff manuals [4].

The normal way (not on embedded compact) is to copy the installation file to the controller and install it like a normal program.

3.3.3. TCP/IP framework Licencing

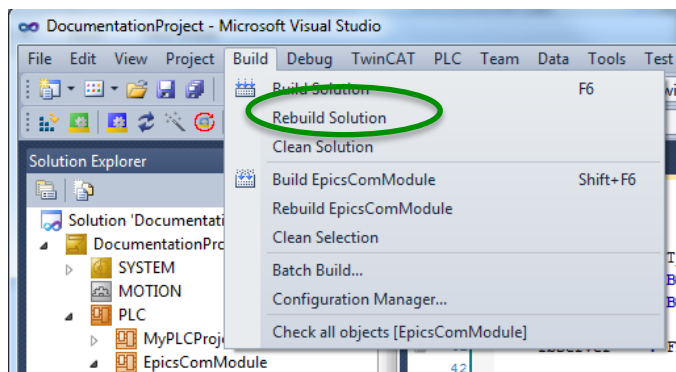
A licence is needed for the TCP/IP communication framework supplied by Beckhoff, which EpicsComModule is based on. **This is not a mandatory step when testing since the Beckhoff engineering software will automatically generate a trial licence for the TCP/IP framework.**

Note: The communication will work with a trial licence for 7 days (without a commercial licence). The trial licence can be regenerated after each 7-day period. Therefore no licence is needed for trying the EpicsComModule. The commercial licence for the TCP-IP framework is handled in the same way as other Beckhoff licences like for example the TwinCAT PLC runtime licence.

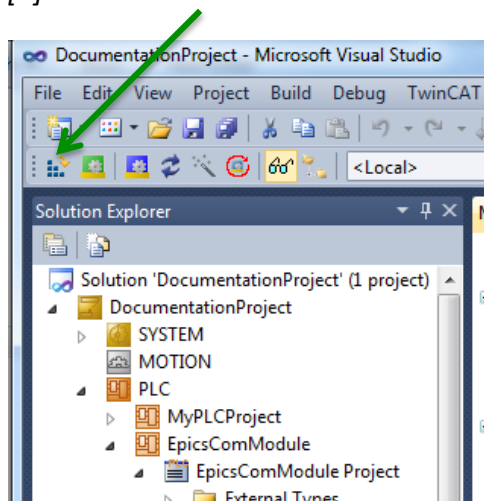
3.3.4. Build and download

The source code needs to be compiled downloaded to the controller.

1. Rebuild solution



2. Activate configuration and download to the TwinCAT controller. For more detailed help regarding compiling, activating configuration and downloading see the Beckhoff help system [2].



3. Press the "Play"-button for both PLC-projects (EpicsComModule and MCAG_PLC_Project) in the MCAG_Base_Project.

4. The MCAG_Base_Project should now be executing on the controller.

3.3.5. Test communication

The connection can be tested with Telnet, see chapter 9.1, "Telnet access".

If the communication is not working then read Chapter 10, "Trouble shooting".

3.4. EpicsComModule Settings

EpicsComModule configurations can be accessed in the “Global_Variables” folder, see Figure 5. The different settings are described in Table 5.

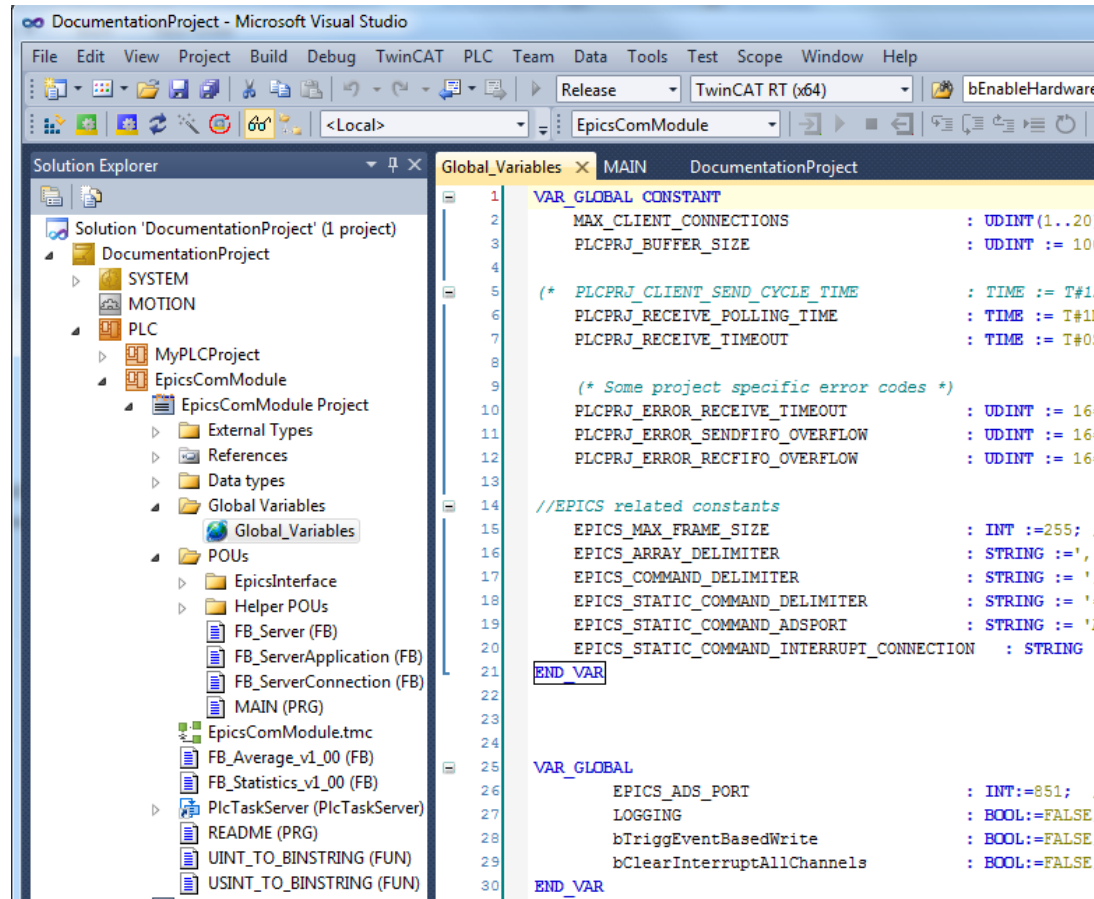


Figure 5: EpicsComModule constants and settings

Table 5: Description of some the EpicsComModule constants and settings

Name	Constant or variable	Default value	Description
MAX_CLIENT_CONNECTIONS	Constant	5	This constant defines the maximum number of client connections
PLCPRJ_BUFFER_SIZE	Constant	20000	Communication buffer size
EPICS_MAX_FRAME_SIZE	Constant	1400	Maximum frame length
EPICS_ARRAY_DELIMITER	Constant	“,”	Character between array element values in a text string
EPICS_COMMAND_DELIMITER	Constant	“;”	Character to mark end of a command

EPICS_STATIC_COMMAND_DELIMITER	Constant	“/”	Character to mark end of a static command that needs parsing
EPICS_STATIC_COMMAND_ADSPORT	Constant	“ADSPORT=”	Command string for temporary assignment of ADS-port number
EPICS_STATIC_COMMAND_INTERRUPT	Constant	“INTERRUPT”	Command string for settings connection type

4. EPICS CONFIGURATION

EPICS utilises the ASYN-driver and Stream Device to output command strings over TCP-IP, both ASYN-driver and Stream Device therefore needs to be installed and configured, see Figure 6.

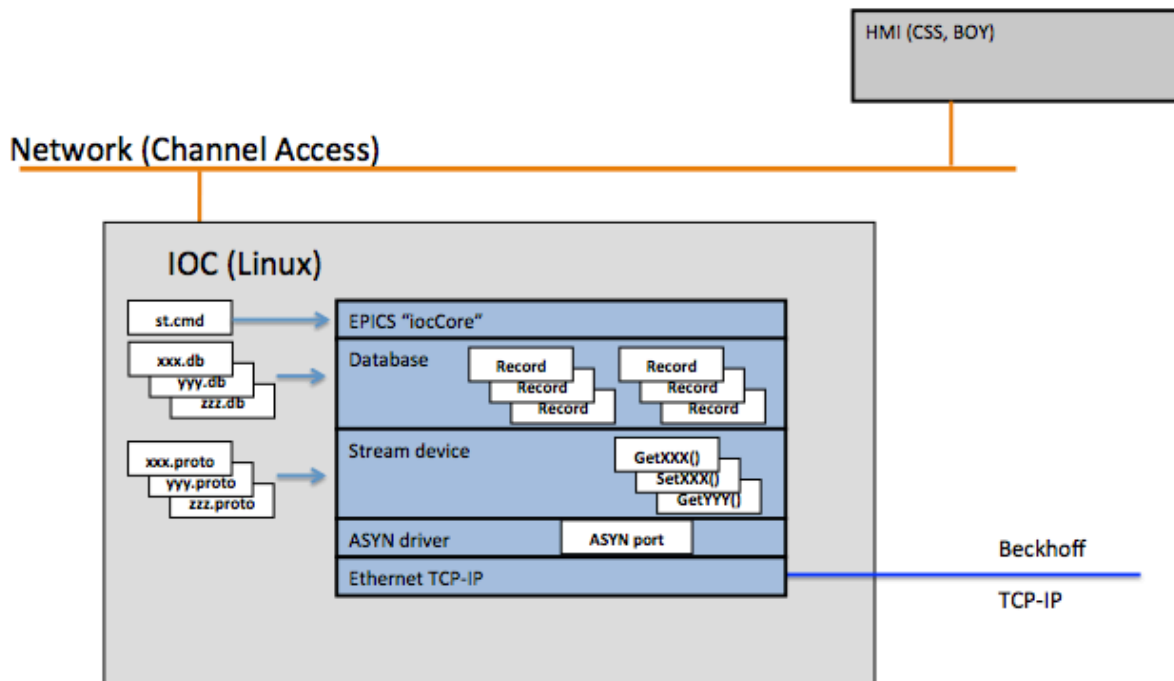


Figure 6: EPICS Communication

Information on how to install/compile EPICS, ASYN-driver and Steam device is accessible on the EPICS online web page [1] and will therefore not be covered in this manual.

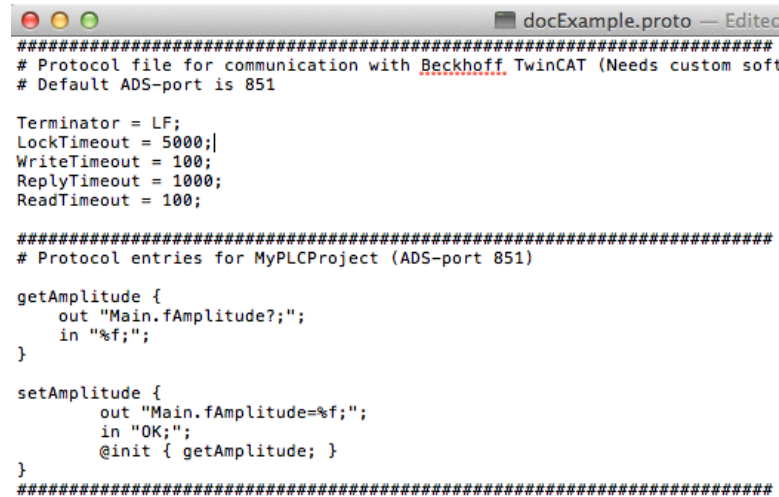
In order to configure EPICS records using stream device, normally three types of files are needed:

- "*.proto"-file for configuration of the communication protocol (formatting of commands and data)
- "*.db"-file for database record configuration
- st.cmd -file which is the start-up script for EPICS

4.1. proto-file

The proto file defines functions for sending, receiving and formatting data, see Figure 7. Detailed information about how to write this file can be accessed on the EPICS web page [1].

Example:



```
#####
# Protocol file for communication with Beckhoff TwinCAT (Needs custom soft
# Default ADS-port is 851

Terminator = LF;
LockTimeout = 5000;
WriteTimeout = 100;
ReplyTimeout = 1000;
ReadTimeout = 100;

#####
# Protocol entries for MyPLCProject (ADS-port 851)

getAmplitude {
    out "Main.fAmplitude?";
    in "%f";
}

setAmplitude {
    out "Main.fAmplitude=%f";
    in "OK";
    @init { getAmplitude; }
}

#####
```

Figure 7: Example of proto-file

In the example above two functions are defined:

- getAmplitude: Reads the variable Main.fAmplitude (default ADS-port)
- setAmplitude: Writes to the variable Main.fAmplitude (default ADS-port)

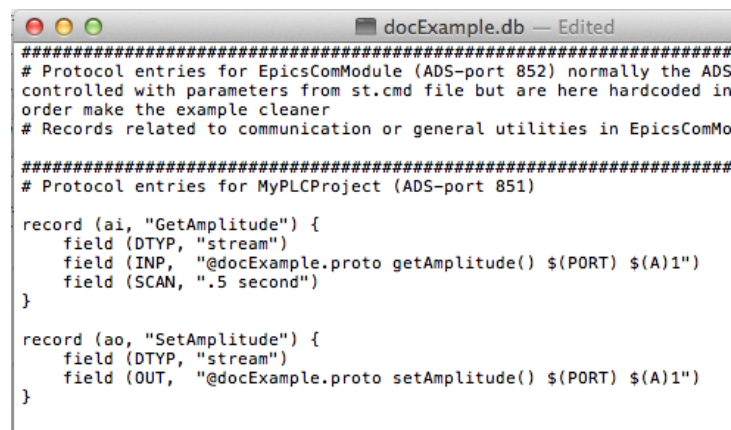
The command text string that will be sent is defined after the “out” statement in each function. The expected returned input string is formatted after the “in” statement in each function. Functions that are configured to set data, “setAmplitude”, can be configured with an initial value. In this case the initial value will be read by the function “getAmplitude”.

Note: The above file is only an example. Templates and variables can be used to make a more flexible and generic architecture.

4.2. db-file

The .db file defines the Epics records, see Figure 8. Detailed information about how to write this file can be accessed on the EPICS web page [1].

Example:



```
#####  
# Protocol entries for EpicsComModule (ADS-port 852) normally the ADS  
# controlled with parameters from st.cmd file but are here hardcoded in  
# order make the example cleaner  
# Records related to communication or general utilities in EpicsComMo  
#####  
# Protocol entries for MyPLCProject (ADS-port 851)  
#####  
record (ai, "GetAmplitude") {  
    field (DTYP, "stream")  
    field (INP, "@docExample.proto getAmplitude() ${PORT} ${A}1")  
    field (SCAN, ".5 second")  
}  
  
record (ao, "SetAmplitude") {  
    field (DTYP, "stream")  
    field (OUT, "@docExample.proto setAmplitude() ${PORT} ${A}1")  
}
```

Figure 8: Example of db-file

In the example above two records are defined:

- GetAmplitude: Analog input record that link to the function “getAmplitude” in the com.proto-file.
- SetAmplitude: Analog output record that link to the function “setAmplitude” in the com.proto-file.

The actual value of the analog input record, “GetAmplitude” will be updated each 0.5 seconds by “executing” the function getAmplitude defined in the docExample.proto-file.

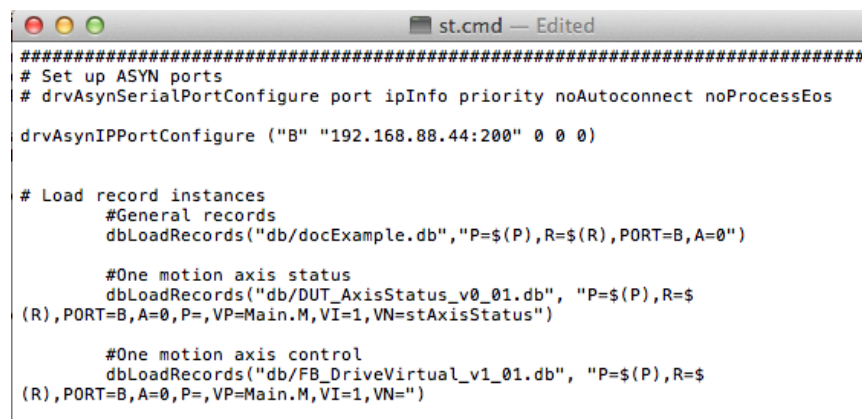
The value of the analog output record “SetAmplitude” will be initialized by executing the “getAmplitude” when the EPICS ioc starts. For each updated value of the record (from HMI, script or EPICS console) the function “setAmplitude” will be “executed” and thereby the new value will be written to the PLC.

Note: The above file is only an example. Templates and variables can be used to make a more flexible and generic architecture.

4.3. st.cmd-file

The st.cmd file can be describes as a start-up file of the EPICS IOC that contains configurations of the EPICS environment and ASYN-driver as well as links to db-files that should be loaded, see Figure 9.

Example:



```
#####
# Set up ASYN ports
# drvAsynSerialPortConfigure port ipInfo priority noAutoconnect noProcessEos

drvAsynIPPortConfigure ("B" "192.168.88.44:200" 0 0 0)

# Load record instances
#General records
dbLoadRecords("db/docExample.db", "P=$(P),R=$(R),PORT=B,A=0")

#One motion axis status
dbLoadRecords("db/DUT_AxisStatus_v0_01.db", "P=$(P),R=$(R),PORT=B,A=0,P=,VP=Main.M,VI=1,VN=stAxisStatus")

#One motion axis control
dbLoadRecords("db/FB_DriveVirtual_v1_01.db", "P=$(P),R=$(R),PORT=B,A=0,P=,VP=Main.M,VI=1,VN=")
```

Figure 9: Example of st.cmd file

drvAsynIPPortConfigure: Configures the ASYN port for TCP/IP communication to a server with IP-adress 192.168.88.44 on port 200.

dbLoadRecords: Loads records from the “docExample.db”, “DUT_AxisStatus_v_01.db” and “FB_DriveVirtual_v1_01.db” files

Note: The above file is only an example. For “DUT_AxisStatus_v_01.db” and “FB_DriveVirtual_v1_01.db” macros are used for the variable names and axis number in order to increase flexibility.

4.4. Supported record types

Normally all records types that are supported by Stream Device and ASYN-driver should be supported by the EpicsComModule, however, not all types have been tested. The tested record types are listed in Table 6.

Table 6: Tested record types

Record type	Epics type	Comment
Binary input	bi	Binary data input
Binary output	bo	Binary data output
Analog input	ai	Analog data input
Analog output	ao	Analog data output
Motor record	Motor Record	Record to handle on axis of motion
Waveform	waveform	Analog array data input/output

4.5. Motor Record Support

Basic functionality of the EPCIS motor record is supported.

4.5.1. EPICS

Information and source of a model 3 driver for the EPICS motor record support for TwinCAT can be found on Github [6].

4.5.2. TwinCAT

The model 3 driver for the EPICS motor record utilizes the same communication protocol as described in this manual. Therefore the EpicsComModule needs to be loaded and configured in the TwinCAT controller.

The actual control of the motion axis is handled in the TwinCAT controller by a function block called FB_DriveVirtual. This block needs to be called for each axis that should be controlled by the motor record. The FB_DriveVirtual function block supports most of the normal motion applications like:

- Absolute positioning
- Relative positioning
- Modulo positioning
- Constant speed
- Homing

- Jogging
- Handling of limit switches
- Synchronization of axis (not supported by motor record)

The motor record driver will access the variables allocated in the FB_DriveVirtual block by their symbolic names and thereby controlling the motion.

Information and source of FB_DriveVirtual can be found on Github [5].

NOTE: Of course the FB_DriveVirtual can be used with “normal” EPICS records like binary inputs/outputs and analog inputs/output records as well. These records can be automatically generated with a software tool, see chapter 7.

EPICS configuration files generated for FB_DriveVirtual can also be accessed in the sample project on git.

5. PERFORMANCE

The communication performance depends on not only the size of the variables transferred but also how many requests the data are divided into. Normally EPICS, for sampled data, must await a reply from the PLC before the next request command will be sent. This will lead to a large amount of idle waiting time. In order to optimize communication throughput structures and arrays can be used. Large quantities of data can then be transferred for a each single command and thereby reducing wait time. Another way to optimize speed is to use interrupt-based communication. Data will then be pushed from the PLC to EPICS without any commands being sent from EPICS.

An example of a tested communication setup is presented in the table below.

Description	Type	Size [byte]	Sample Rate [Hz]	Total size [kbyte/s]
1 Analog array	ARRAY[1000] OF LREAL	1000*8	10	80
5 Motion Axis	DUT_AxisStatus	88	10	4.4
Hardware status	BOOL	10	10	0.1
General Communication	BOOL, INT, UDINT, LREAL	100	10	1

NOTE: The Communication performance is also depending on the hardware used.

6. RESTRICTIONS

6.1. Communication buffer size

The communication buffer needs to be large enough to handle the communication needs in the application. Buffer size settings can be changed in the global variables folder in the EpicsComModule, Figure 10.

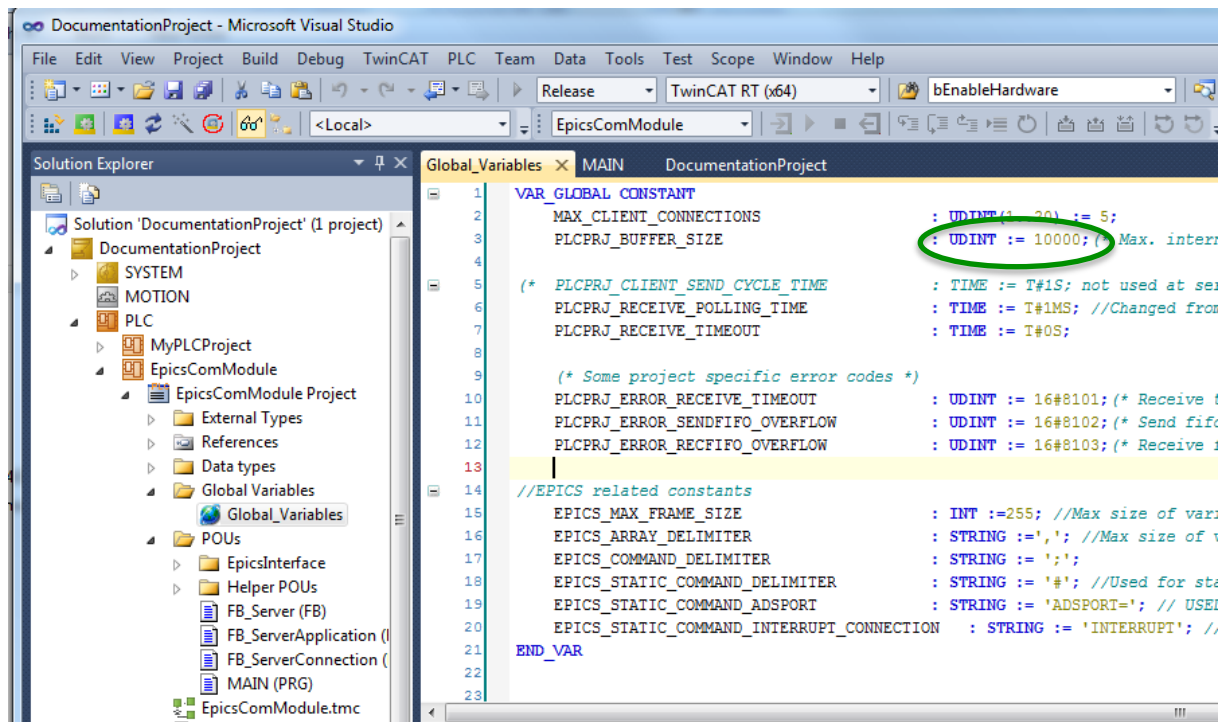


Figure 10: Communication buffer size setting

7. EPICS CONFIGURATION FILE GENERATION TOOL

An tool to generate EPICS configuration files (*.proto and *.db files) from defined TwinCAT 3 structures (DUT) or function blocks (FB) have been developed, see Figure 11 and Figure 12. The tool can open TwinCAT structures or function blocks and generate EPICS configuration files for selected variables. The tool also allows optimization for use of macros on the EPICS side.

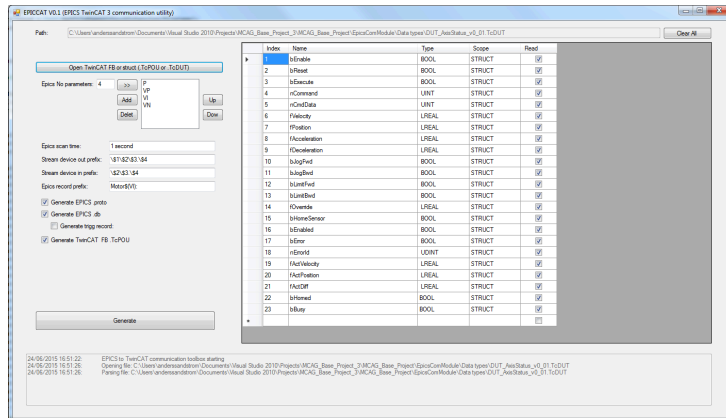


Figure 11: EPICS TwinCAT 3 communication utility (structures)

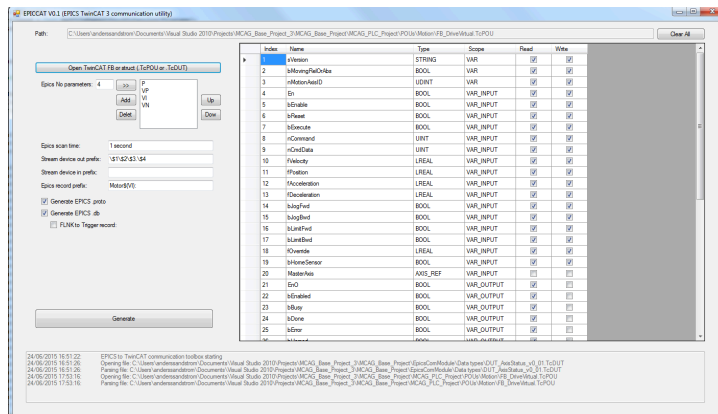


Figure 12: EPICS TwinCAT 3 communication utility (function block)

A separate manual will be generated for this tool.

8. FUTURE WORK/IDEAS

8.1. Interrupt based communication

The current implementation of interrupt based communication needs to be partly hardcoded in the EpicsComModule. A better solution would be to be able to configure from EPICS which variables to subscribe to and at which conditions they should be sent (data only sent to EPICS when they change value over a certain threshold) resulting in reduced communication load and therefore increased performance.

NOTE: Of course the hardcoded option should always be available since this will be the fastest way to communicate (shortest reaction time).

Status: This functionality is currently under development.

8.2. Lookup table for addresses

In order to increase of communication even more a lookup table for the symbolic variable names and the corresponding addresses and types could be implemented in the PLC. The EpicsComModule would then be able to answer a request faster.

Status: This functionality is currently under development.

8.3. Binary communication

Another way to further increase communication speed would be to introduce binary communication instead of ASCII strings. Most data types would require less communication space if transferred binary.

Status: This functionality is tested but some issues were identified at the EPICS side. Further work needed.

8.4. Source code on Github

Source code for the EpicsComModule, a complete sample project, and Motor record support are accessible on Github. The source code for the utility program is not yet on Github.

8.5. Optimization of memory usage and performance

Optimization of the code will be done consciously.

9. TIPS AND TRICKS

9.1. Telnet access

The default protocol allows for Telnet communication to the TwinCAT controller, Figure 13. This can be useful for debugging or development of custom applications.

The syntax for connecting with Telnet is normally: “telnet IP-address port”.



```
anderssanmbp:~ anderssandstrom$ telnet 192.168.88.49 200
Trying 192.168.88.49...
Connected to 192.168.88.49.
Escape character is '^]'.
Main.M1.bEnable?
0;
Main.M1.fVelocity=100;
OK;
Main.M1.fVelocity?
100.0;
█
```

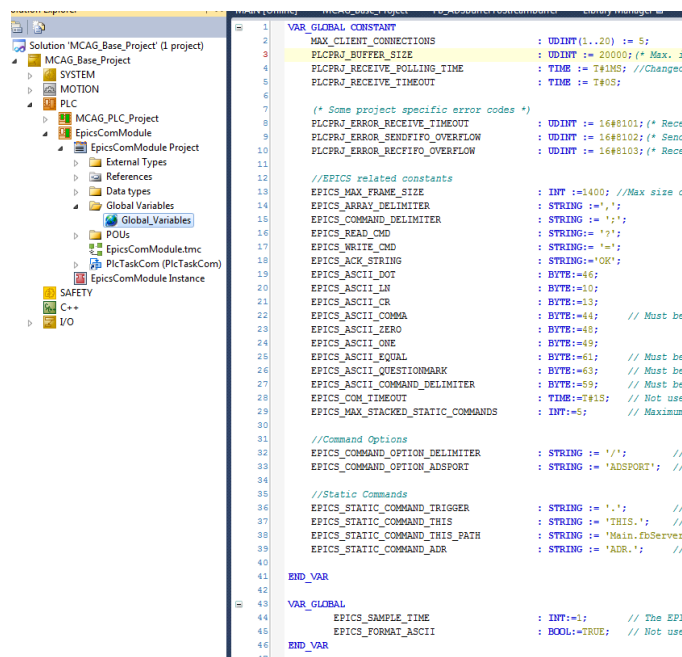
Figure 13: Telnet access

NOTE: The default port used for the connection is 200.

9.2. Customization of the protocol

9.2.1. General customization settings

Most parts of the protocol can be redefined and customized by changing global constants in the EpicsComModule project, see chapter 0.



```
1  VAR_GLOBAL CONSTANTS
2  MAX_CLIENT_CONNECTIONS : UDINT(1..20) := 5;
3  MAX_BUFFER_SIZE         : UDINT := 20000; (* Max. i
4  PLCPRJ_RECEIVE_POLLING_TIME : TIME := T#1MS; //Changed
5  PLCPRJ_RECEIVE_TIMEOUT    : TIME := T#0S;
6
7  (* Some project specific error codes *)
8  PLCPRJ_ERROR_RECEIVE_TIMEOUT : UDINT := 16#0101; (* Rece
9  PLCPRJ_ERROR_SENDTIFO_OVERFLOW : UDINT := 16#0102; (* Send
10 PLCPRJ_ERROR_RECVTIFO_OVERFLOW : UDINT := 16#0103; (* Rece
11
12 //EPICS related constants
13 EPICS_MAX_FRAME_SIZE : INT := 1400; //Max size c
14 EPICS_ARRAY_DELIMITER : STRING := ',';
15 EPICS_COMMAND_DELIMITER : STRING := ':';
16 EPICS_READ_CMD : STRING := '?';
17 EPICS_WRITE_CMD : STRING := '=';
18 EPICS_ACK_STRING : STRING := 'OK';
19 EPICS_ASCII_CR : BYTE := 13;
20 EPICS_ASCII_LN : BYTE := 10;
21 EPICS_ASCII_CR : BYTE := 13;
22 EPICS_ASCII_COMMA : BYTE := 44; // Must be
23 EPICS_ASCII_ZERO : BYTE := 48;
24 EPICS_ASCII_ONE : BYTE := 49;
25 EPICS_ASCII_EQUAL : BYTE := 61; // Must be
26 EPICS_ASCII_QUESTIONMARK : BYTE := 63; // Must be
27 EPICS_ASCII_COMMAND_DELIMITER : BYTE := 59; // Must be
28 EPICS_CMD_TIMEOUT : TIME := T#1S; // Not use
29 EPICS_MAX_STACKED_STATIC_COMMANDS : INT := 5; // Maximum
30
31 //Command Options
32 EPICS_COMMAND_OPTION_DELIMITER : STRING := ':'; //
33 EPICS_COMMAND_OPTION_ADSPORT : STRING := 'ADSPORT'; //
34
35 //Static Commands
36 EPICS_STATIC_COMMAND_TRIGGER : STRING := 'T'; //
37 EPICS_STATIC_COMMAND_THIS : STRING := 'THIS'; //
38 EPICS_STATIC_COMMAND_THIS_PATH : STRING := 'Main.fbServer';
39 EPICS_STATIC_COMMAND_ADR : STRING := 'ADR.'; //
40
41 END_VAR
42
43 VAR_GLOBAL
44 EPICS_SAMPLE_TIME : INT := 1; // The EPI
45 EPICS_FORMAT_ASCII : BOOL := TRUE; // Not use
46
47 END_VAR
```

Figure 14: EpicsComModule constants

9.2.2. Settings for individual connection

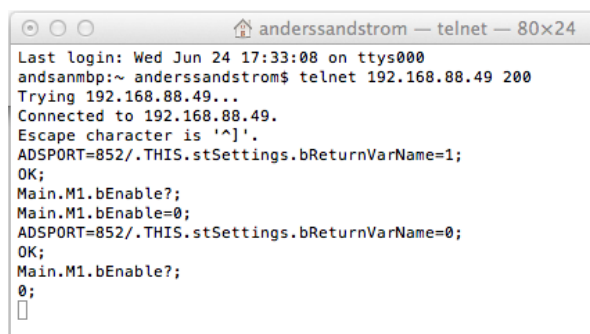
Some settings can be made locally on the individual connection. These settings can be accessed by the “.THIS.” command.

9.2.2.1. Return of variable name

The variable name can be returned together with data if the setting “bReturnVarName” is enabled for the connections. The following command enables this functionality:

```
“ADSPORT=852/.THIS.stSettings.bReturnVarName=1;”
```

Figure 15 shows the difference between the two modes.



```
anderssandstrom — telnet — 80x24
Last login: Wed Jun 24 17:33:08 on ttys000
andsanmbp:~ anderssandstrom$ telnet 192.168.88.49 200
Trying 192.168.88.49...
Connected to 192.168.88.49.
Escape character is '^]'.
ADSPORT=852/.THIS.stSettings.bReturnVarName=1;
OK;
Main.M1.bEnable?;
Main.M1.bEnable=0;
ADSPORT=852/.THIS.stSettings.bReturnVarName=0;
OK;
Main.M1.bEnable?;
0;
□
```

Figure 15: Setting: Return variable name

9.2.2.2. Return data on write

Data can be returned also for write commands. The EpicsComModule will then first write the data and then read the data from the symbolic variable or absolute address. The following command enables this functionality:

```
“ADSPORT=852/.THIS.stSettings.bReturnData=1;”
```

Figure 16 shows the difference between the two modes.



```
anderssandstrom — telnet — 80x24
Last login: Wed Jun 24 18:09:56 on ttys000
andsanmbp:~ anderssandstrom$ telnet 192.168.88.49 200
Trying 192.168.88.49...
Connected to 192.168.88.49.
Escape character is '^]'.
ADSPORT=852/.THIS.stSettings.bReturnData=1;
1;
Main.M1.fVelocity=1000;
1000.0;
ADSPORT=852/.THIS.stSettings.bReturnData=0;
OK;
Main.M1.fVelocity=10;
OK;
□
```

Figure 16: Setting: Return data on write

NOTE: The written and read data are not compared in the EpicsComModule.

9.2.2.3. Change of default ADS-Port

The default ADS-port for communication is normally set to 851 which corresponds to the first PLC project in the TwinCAT system. The default ADS-port can be changed by writing to the default ADS-Port variable:

Example (set default ADS-port for the current connection to 852):

```
"ADSPORT=852/.THIS.stSettings.nADSPort=852;"
```

Example (read default ADS-port for the current connection):

```
"ADSPORT=852/.THIS.stSettings.nADSPort?;"
```

NOTE: At PLC reboot the default ADS-port will be reset to its initiation value which normally is 851. The change of ADS-port command therefore needs to be sent at reconnection to the TwinCAT system.

9.2.2.4. Read Connection Index

Each connection will be handled through a function block, FB_ServerApplication, in the EpicsComModule PLC. The default setting is that 5 connections are allowed and allocated in the PLC as an array of FB_ServerApplication. For debug purpose it could be interesting to know which of these connections you are currently connected to. The following command will return the index of the connection you are currently connected:

```
"ADSPORT=852/.THIS.stSettings.nConnectionIndex?;"
```

NOTE: This variable is read only.

9.3. Change of default ADS-Port address

The default ADS-Port can be changed for all the connections by modifying the structure DUT_ComSettings in the EpicsComModule, see **Error! Reference source not found..** This setting will also be loaded as initial value after PLC reboot.

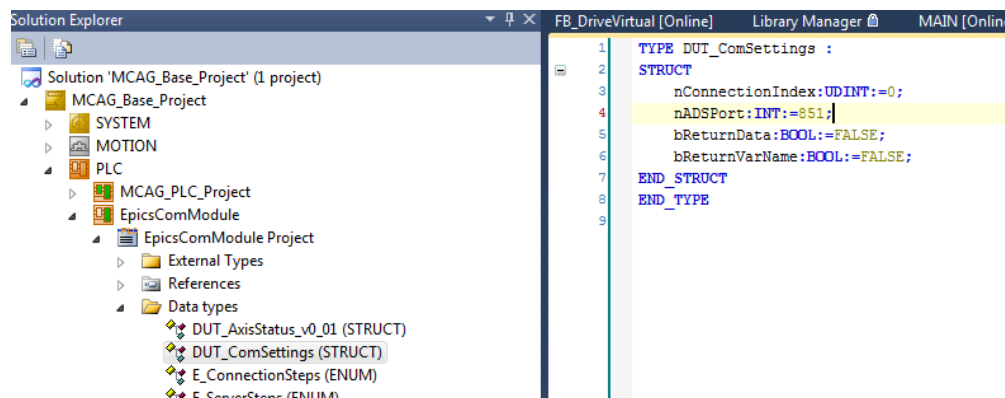


Figure 17: Change of default ADS-Port for all connections

9.4. Identify to which motion axis the FB_DriveVirtual is connected

Normally when running a motion in a TwinCAT system the hardware drive is linked to a Motion Axis within the TwinCAT motion system and then this Motion Axis is linked to a PLC program running motion functions like FB_DriveVirtual.

However several parameters can be set directly on the Motion Axis: Maximum speed, acceleration, scaling's, homing procedures, and more, Figure 18:

- “Axis n” for general settings of the axis
- “Enc” for encoder parameters (could be several)
- “Drive” for drive settings
- “Ctrl” for controller settings

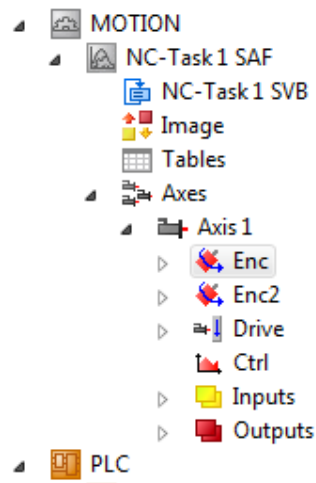


Figure 18: Motion Axis Parameters

These parameters can be accessed using the “.ADR.” command using absolute addresses described in chapter 2.5.2. The absolute addresses are allocated according to a system were ranges are defined for each sub section to the axis, Table 7.

Table 7: Index Group base addresses for Axis object

Group	IndexGroup Base address
Axis	16#4000
Enc	16#5000
Drive	16#7000
Control	16#6000

The axis index is then used to as an offset to the IndexGroup address. The Axis parameters for motion axis 1 will thereby be accessed by using the IndexGroup $16\#4000+1=16\#4001$ ($16\#4002$ for axis 2).

The motion axis number connected to certain FB_DriveVirtual function block can be accessed from the PLC by reading the nMotionAxisID variable in FB_DriveVirtual, Figure 19.

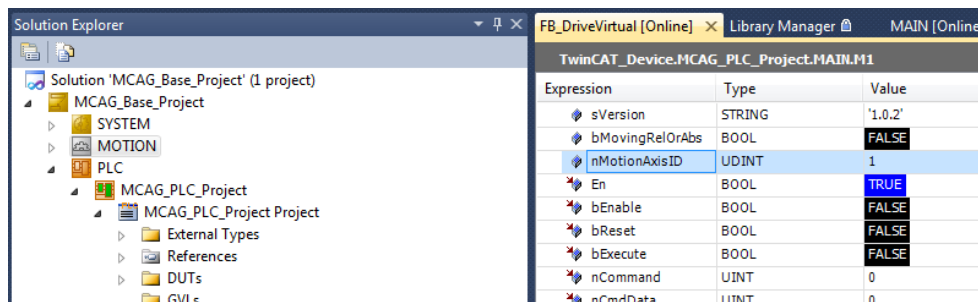


Figure 19: nMotionAxisID variable in FB_DriveVirtual

Example (the instance of FB_DriveVirtual is called M1):

“Main.M1.nMotionAxisID?;”

Addresses for settings of the motion axis connected to FB_DriveVirtual can then be calculated.

Example: Set the “Maximum Velocity” parameter to 100.0 for the axis linked to M1 (instance of FB_DriveVirtual):

Step 1: *“Main.M1.nMotionAxisID?;” Returns “1;”*

Step 2: *Calculate IndexGroup address=16#4000+16#1=16#4001.*

Step 3: *Identify IndexOffset, size and type from engineering tool or manual= 16#27,8,5 (see chapter 2.5.2.1 for details)*

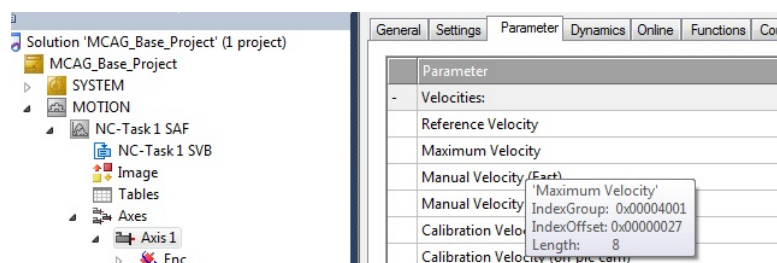


Figure 20: Absolute address of axis 1

Step 4: *Run ADR command (to ADSPORT 501):*

“ADSPORT=501/.ADR.16#4001,16#27,8,5=100;”

NOTE: Normally the link between a Motion Axis and PLC code can be fixed in the TwinCAT solution/project (M1 always links to Axis 1). However, in order to ensure that always the correct parameters are accessed the nMotionAxisID should always be read from FB_DriveVirtual (it is also possible to relink axis from software resulting in that the axis link can change at runtime).

9.4.1. Identify ADS-Ports for PLC-Modules

The ADS-port numbers assigned to your PLC project and the EpicsComModule can be accessed by double clicking the projects (in this example 851 for MyPLCProject and 852 for EpicsComModule), see Figure 21.

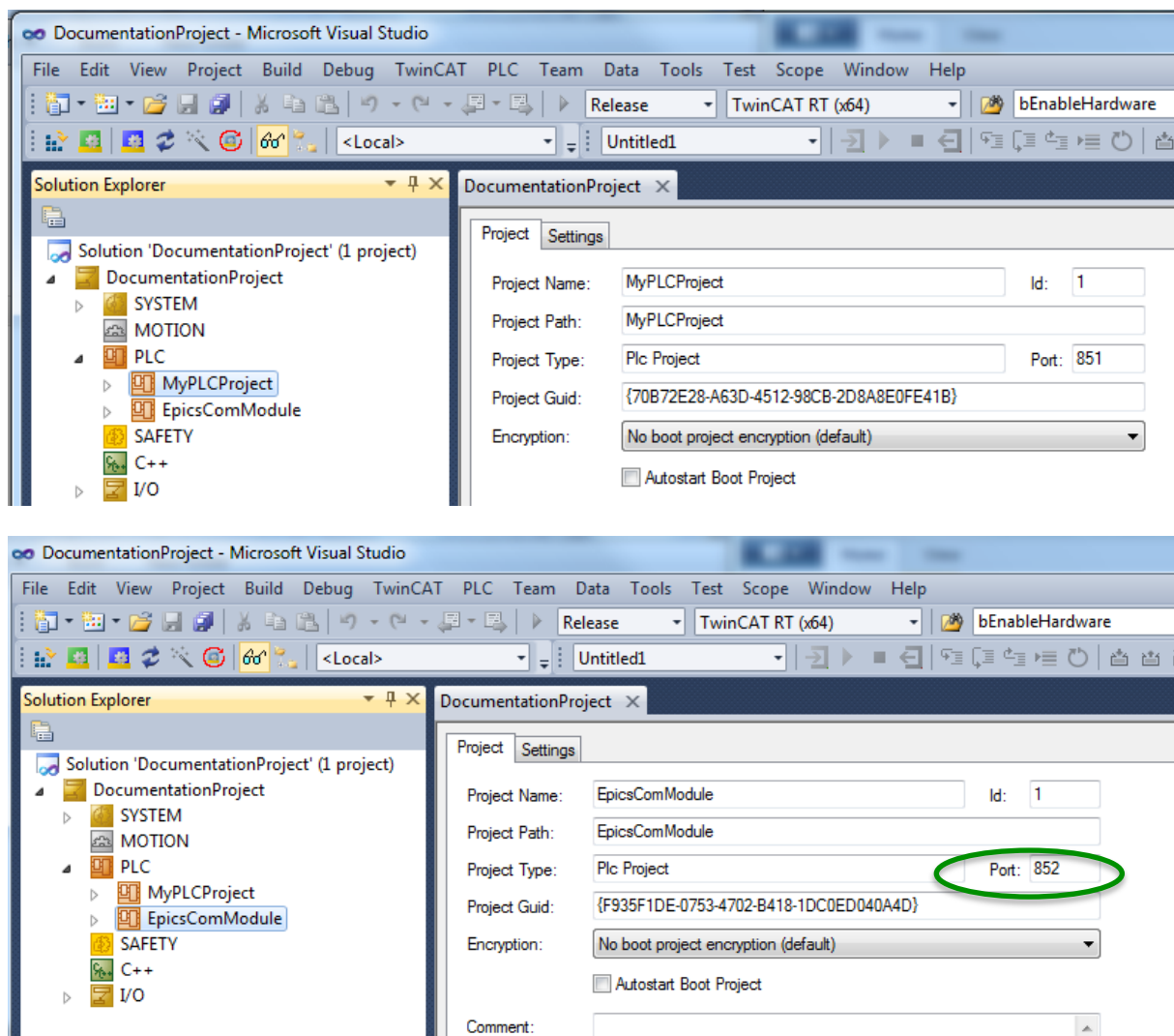


Figure 21: ADS-port number for PLC-projects

10. TROUBLE SHOOTING

This chapter list some common problems and their solutions.

10.1.1. ADS error 1797

Problem:

ADS error 1797 is returned.

Reason:

An “ADS error 1797” can be generated if an accessed symbolic variable in TwinCAT system have a comment longer that 255 characters. The EpicsComModule asks the ADS-router for the absolute address of the symbolic address and will also receive the comment associated with the symbolic variable. A string of maximum 255 byte is allocated for the comment.

Solution:

Keep the comments to variables shorter than 255 characters.

10.1.2. Will not accept connections

Problem:

Connection problems to EpicsComModule

Reason 1:

Could be related to that the Beckhoff TCP/IP framework have not been installed on the controller.

Solution 1:

Download and install the Beckhoff TCP/IP framework.

Reason 2:

Could be related to that the TCP/IP server licence is missing or the trial licence has expired.

Solution 2:

The TCP/IP trial license will not regenerate automatically. The TCP/IP server license (TF6310) needs to be selected as project license in the three structure: system->license.

Reason 3:

The controller is not executing.

Solution 3:

Ensure that the correct software is loaded and that it is executing.

10.1.3. Library mismatch

Problem:

When opening a TwinCAT solution or project in the engineering tool (based on Visual Studio) an error message about library mismatch is displayed and the project will not compile.

Reason:

The project and code is developed in an other version of engineering tool and therefore have access to different versions of standard libraries.

Solution

Normally the error message concerns version conflicts of the library's "TC2_EtherCat" or "TC2_MC2". New references to the accessible library's needs to be created and the links to the library's that don't exists needs to be deleted for each PLC-module, Figure 22. After changing a library the functionality of the PLC-code should be tested thoroughly.

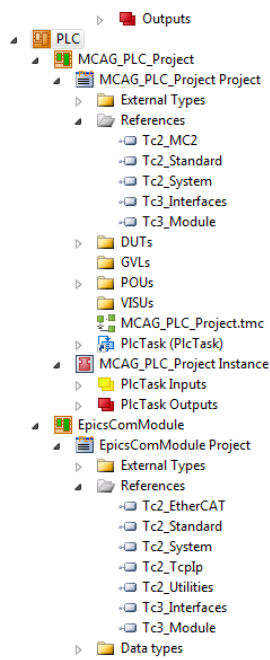


Figure 22: Update of references

10.1.4. Performance issues

Problem:

The EpicsComModule loads the CPU too much.

Reason:

The EpicsComModule can run a range of different CPU:s with different performance. Sometimes, on low performance CPU:s some settings needs/ can be changed in order to reduce the CPU-load.

Solution:

- Set a low priority (high value) on the task that executes the EpicsComModule (default is 26)
- If possible use a dedicated CPU core for the communication.
- Reduce the base time setting to 1ms for slower systems, Figure 23.

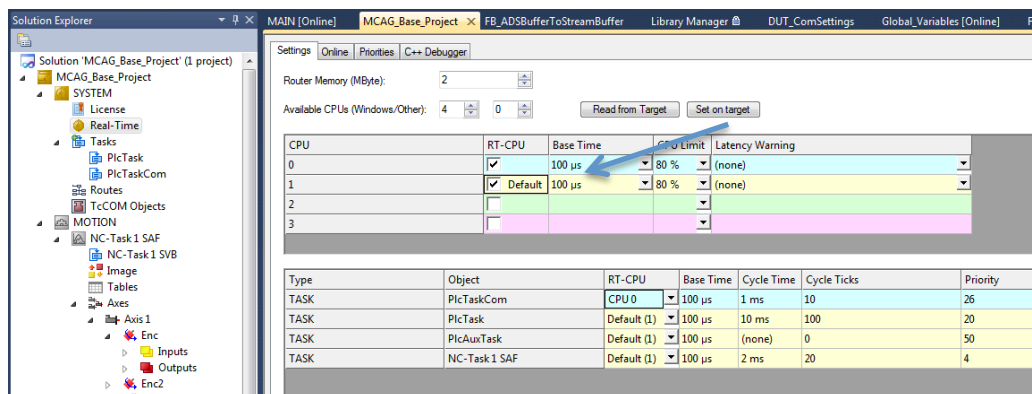


Figure 23: Real-Time settings

- Reduce buffer sizes in the EpicsComModule, Figure 24. The default value is 20kb and this number can normally be reduced rather much (5000 is enough for most use-cases).

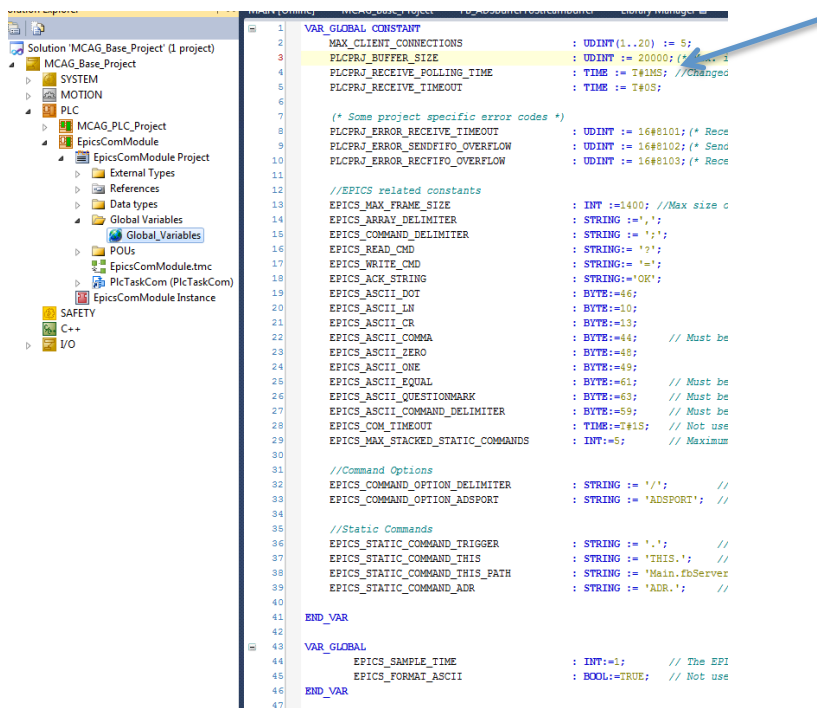
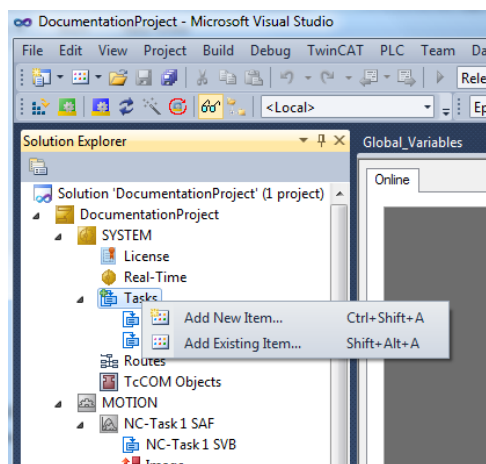


Figure 24: EpicsComModule buffer settings

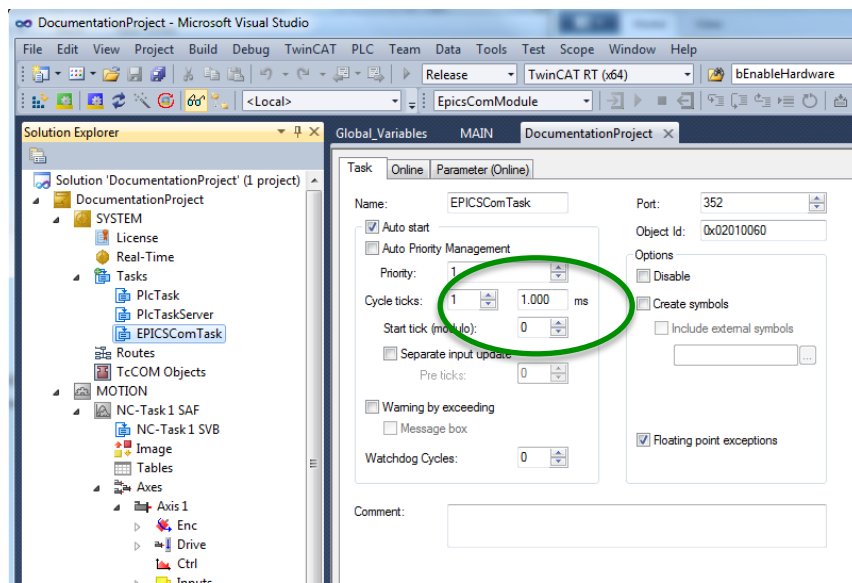
10.1.5. Create and link to new Task

The EpicsComModule Main program is normally linked to a TwinCAT task in the MCAG_Base_Project. However, if this link is lost the following procedure needs to be followed:

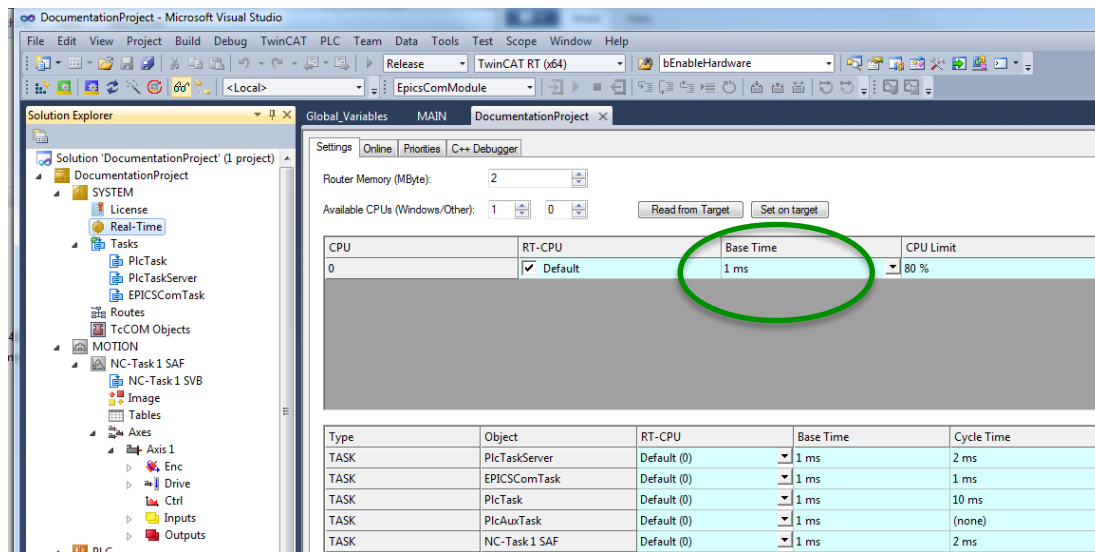
1. Create a new task by right clicking on the Systems->Task folder and choose “Add New Item”.



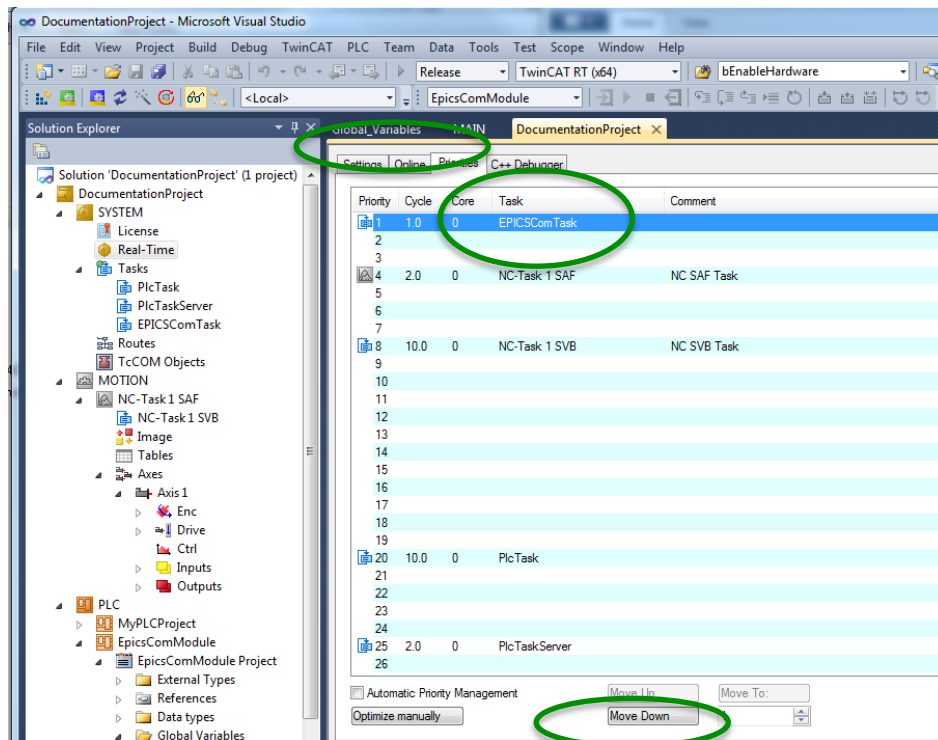
2. Change cycle ticks to 1 ms for low performing hardware and to 0.1ms for high performance controllers / high communication needs.



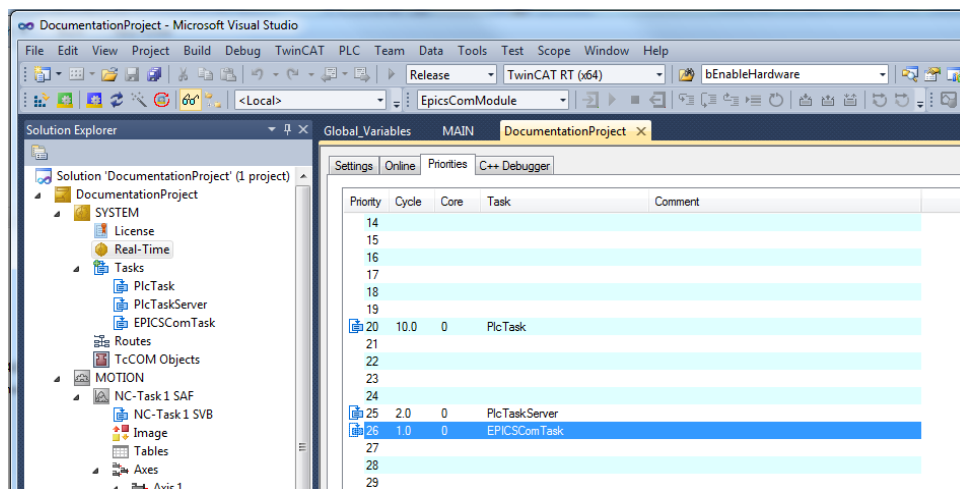
Note: In order to set the cycle ticks time to a lower value than 1 ms, the base time needs to be changed to a lower value, 50microseconds. The base time can be changed under Systems->Real-Time.



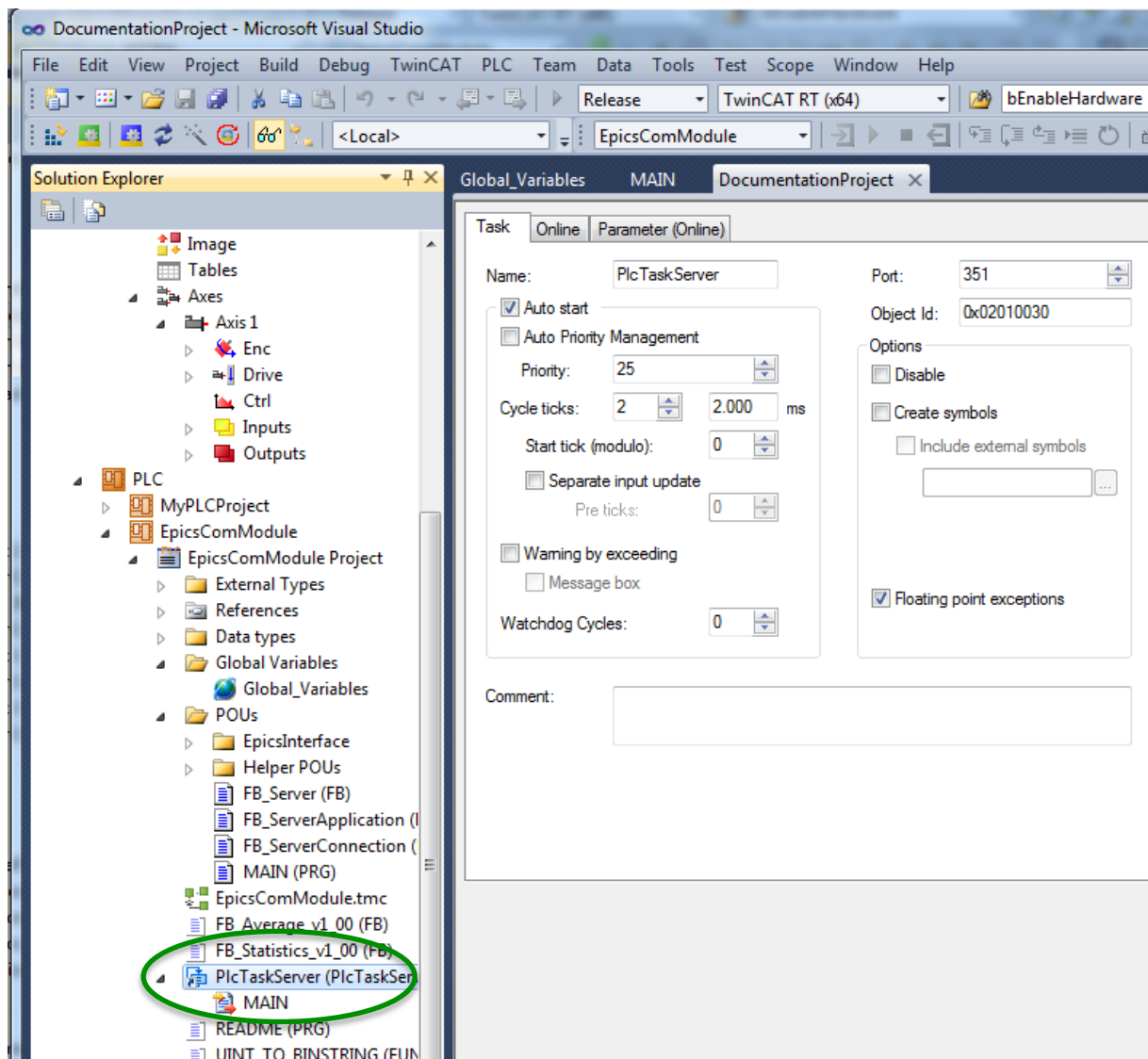
3. Set a lower priority of the communication task. Double click on the System->Real-Time folder and choose the "Priorities" tab. Choose the newly created task and press the "Move Down" button until the priority of the task is higher than 25 (higher number means lower priority).



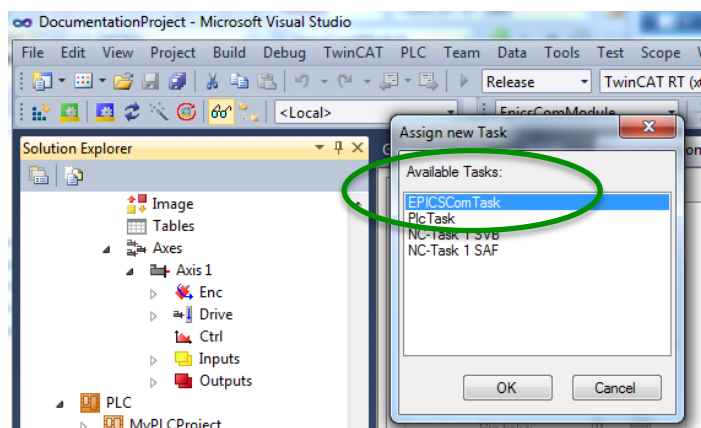
The priorities list should now look like below:



4. Link EpicsComModule Main program to the Task. Right click on "PLC->EpicsComModule->PLCTaskServer(PLCTaskServer)" and choose "Assign to Task"



5. Choose the created Task and press "OK" button.



6. Rebuild and download the configuration

11. REFERENCES

1. EPICS, Experimental Physics and Industrial Control System:
www.aps.anl.gov/epics/
2. Beckhoff Infosys: www.infosys.beckhoff.com
3. Motion Control and Automation Group: Manual for standard Function Block
FB_DriveVirtual
4. Beckhoff download web site: www.beckhoff.com
5. TwinCAT MCAG_Base_Project (EpicsComModule and FB_DriveVirtual) on Github:
6. Motor Record support for TwinCat on Github:

12. APPENDIX 1: SUPPORTED STRUCTURES

12.1. DUT_AxisStatus

The structure DUT_AxisStatus includes all information available for one motion axis and can be **read** with one command. The structure is normally used together with a TwinCAT function block FB_DriveVirtual that controls motion for one axis.

The DUT_AxisStatus structure contains the following data:

	Name	Data type
1	bEnable	BOOL
2	bReset	BOOL
3	bExecute	BOOL
4	nCommand	UINT
5	nCmdData	UINT
6	fVelocity	LREAL
7	fPosition	LREAL
8	fAcceleration	LREAL
9	fDeceleration	LREAL
10	bJogFwd	BOOL
11	bJogBwd	BOOL
12	bLimitFwd	BOOL
13	bLimitBwd	BOOL
14	fOverride	LREAL
15	bHomeSensor	BOOL
16	bEnabled	BOOL
17	bError	BOOL
18	nErrorId	UDINT
19	fActVelocity	LREAL
20	fActPosition	LREAL
21	fActDiff	LREAL
22	bBusy	BOOL
23	bHomed	BOOL

Example:

EPICS Command string: *"Main.M1.stAxisStatus?;"*

EpicsComModule will return: *"Main.M1.stAxisStatus: 1,0,1,2,;"*

12.2. DUT_HardwareStatus

TODO. Under development (supported in older versions)

12.3. DUT_ChopperAxisStatus

TODO. Under development (supported in older versions)

12.4. DCTIMESTRUCT

TODO. Under development (supported in older versions)

12.5. DUT_Stats

TODO. Under development (supported in older versions)

12.6. DUT_HardwareStatus_32Bits_v0_01

TODO. Under development (supported in older versions)

12.7. DUT_HardwareStatus_32Float_v0_01

TODO. Under development (supported in older versions)

12.8. DUT_ResolverCalib

TODO. Under development (supported in older versions)

13. APPENDIX 2: QUICK START GUIDE

13.1. TwinCAT

1. Get a copy of the source code from Github:

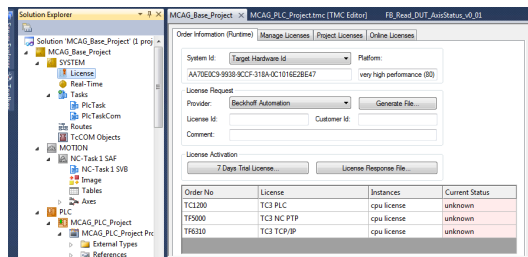
https://github.com/EuropeanSpallationSource/MCAG_Base_Project

(git clone https://github.com/EuropeanSpallationSource/MCAG_Base_Project.git)

A configured TwinCAT project including all needed source should be accessible on a branch called “MCAG_Base_Project_Single_Axis”

2. Download the Beckhoff TCP/IP service “framework” from www.beckhoff.com and install it on the controller.

3. Ensure that the correct trial licences are generated:



NOTE: You may need to choose the TCP/IP licence as “Project Licence” and then press “7-Days Trial licence”-button.

4. Make sure the PLC modules (EpicsComModule and MCAG_PLC_Project) are running by auto start at boot or manual start on the “Play” button.

5. Test communication by connecting to the controller over Telnet:

```
anderssandstrom — telnet — 80x24
andsanmbp:~ anderssandstrom$ telnet 192.168.88.49 200
Trying 192.168.88.49...
Connected to 192.168.88.49.
Escape character is '^'.
Main.M1.bEnable?;
0;
Main.M1.fVelocity=100;
OK;
Main.M1.fVelocity?;
100.0;

```

13.2. EPICS

In order to get Motor Record support the EPICS model 3 motor record driver needs to be used

Get a copy of the source code from Github:

https://github.com/EuropeanSpallationSource/ESS_MCAGmotor

Document Number 1
Date 2015-06-25

(git clone https://github.com/EuropeanSpallationSource/ESS_MCAGmotor.git)