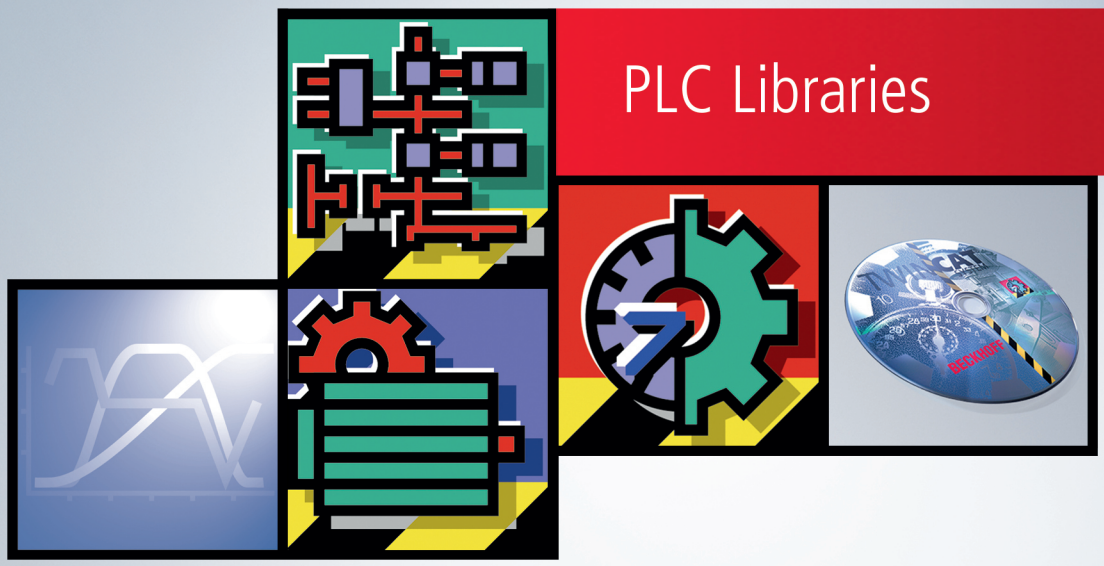
BECKHOFF New Automation Technology

Manual | EN

TX1200

TwinCAT 2 | PLC Library: TcSystem



2022-10-11 | Version: 1.1

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

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

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

Not all those function blocks and functions that are often needed in PLC applications are standardized in IEC61131-3. The system library contains such functions and function blocks for the TwinCAT system which do not belong to the standard scope of IEC61131-3, and which are therefore manufacturer-specific.

Function blocks

ADS function blocks

Name	Description
ADSREAD [▶ 14]	Reading data via ADS.
ADSWRITE [▶ 15]	Writing data via ADS.
ADSRDWRT [▶ 17]	Reading and writing data via ADS.
ADSRDSTATE [▶ 19]	Read the state of a device via ADS.
ADSWRTCTL [▶ 20]	Write control words to a device via ADS.
ADSRDDEVINFO [▶ 22]	Read device information via ADS.
ADSREADEX [▶ 23]	Reading data via ADS and returning the number of read data bytes.
ADSRDWRTEX [▶ 25]	Writing data via ADS and reaturning the number of read data bytes

Extended ADS function blocks

Name	Description
ADSREADIND [▶ 27]	ADSREAD-Indication.
ADSWRITEIND [▶ 28]	ADSWRITE-Indication
ADSRDWRTIND [▶ 29]	ADSRDWRT-Indication
ADSREADRES [▶ 31]	ADSREAD-Response
ADSWRITERES [▶ 32]	ADSWRITE-Response
ADSRDWRTRES [▶ 33]	ADSRDWRT-Response

General function blocks

Name	Description
DRAND [▶ 39]	Random number generator.
GETCURTASKINDEX [▶ 39]	Determines the index of the current task.

Time function blocks

Name	Description
GETSYSTEMTIME [▶ 40]	Read the operating system time stamp.
GETTASKTIME [▶ 41]	Read the target start time of the task.
GETCPUCOUNTER [▶ 42]	Read the CPU cycle counter.
GETCPUACCOUNT [▶ 42]	Read the PLC task cycle counter.

Watchdog function blocks

Name	Description
FB_PcWatchdog [▶ 43]	Activate or Deactivate the PC Watchdog. **Is only available on IPCs with the mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051 !**

F unction blocks for data access

The function blocks can be used to process files from the PLC locally on the PC. The TwinCAT target system is identified by the AMS network address. This mechanism makes it possible, amongst other things, to store or to edit files on other TwinCAT systems in the network. Access to files consists of three sequential phases:

1. Opening the file.
2. Read or write access to the opened file.
3. Closing the file.

Opening the file has the purpose of establishing a temporary connection between the external file, whose name is all that initially is known, and the running program. Closing the file has the purpose of indicating the end of the processing and placing it in a defined output state for processing by other programs.

Name	Description
FB_FileOpen [▶ 50]	Open a file.
FB_FileClose [▶ 46]	Close a file.
FB_FileWrite [▶ 60]	Write to a file.
FB_FileRead [▶ 54]	Read from a file.
FB_FileSeek [▶ 57]	Move the file pointer.
FB_FileTell [▶ 59]	Get the file pointer position.
FB_FilePuts [▶ 53]	Put string to a file.
FB_FileGets [▶ 49]	Get string from a file.
FB_EOF [▶ 45]	Check the end of file.
FB_FileDelete [▶ 47]	Delete a file.
FB_FileRename [▶ 56]	Rename a file.
FB_CreateDir [▶ 65]	Create new directory.
FB_RemoveDir [▶ 67]	Remove directory.

TwinCAT 2.7 function blocks for data access

These function blocks are contained only for compatibility reasons in the library and should not not be used in new projects.

Name	Description
FILEOPEN [▶ 68]	Open a file.
FILECLOSE [▶ 70]	Close a file.
FILEWRITE [▶ 71]	Write to a file.
FILEREAD [▶ 72]	Read from a file.
FILESEEK [▶ 74]	Move the file pointer.

IEC SFC functions

These functions / function blocks are required to use IEC steps or SFC flags in SFC programs /projects.

Name	Description
SFCActionControl [▶ 75]	Enables the using of IEC steps.

Name	Description
AnalyzeExpression [▶ 75] AnalyzeExpressionCombined [▶ 76]	Is required for using SFC flags.
AppendErrorString [▶ 76]	Is required for using SFC flags, to format strings with error description.

Event logger function blocks

The TwinCAT Eventlogger manages all occurring messages (events) in the TwinCAT System. It transfers the data and writes them into the TwinCAT log file.

Name	Description
ADSLOGEVENT [▶ 77]	Sending and acknowledgement of messages to the TwinCAT Eventlogger.
ADSCLEAREVENTS [▶ 80]	Sending and acknowledgement of messages to the TwinCAT Eventlogger.

For further information see the TwinCAT Event Logger documentation.

Functions

General functions

Name	Description
F_SplitPathName [▶ 82]	Splits a path name into components.
F_CreateIPv4Addr [▶ 83]	Converts address bytes to IPv4 address string.
F_ScanIPv4AddrIds [▶ 83]	Converts IPv4 address string to address bytes..
SETBIT32 [▶ 84]	Sets a bit in a DWORD.
CSETBIT32 [▶ 85]	Sets/resets a bit in a DWORD.
GETBIT32 [▶ 86]	Reads a bit from a DWORD.
CLEARBIT32 [▶ 86]	Clears a bit in a DWORD.
LPTSIGNAL [▶ 87]	Outputs a signal on one of the parallel port pins.
F_GetStructMemberAlignment [▶ 88]	Reads data struct member alignment information.
F_GetVersionTcSystem [▶ 90]	Returns the library version info

I/O port access

Name	Description
F_IOPortRead [▶ 91]	Reads from I/O Port
F_IOPortWrite [▶ 91]	Writes to I/O Port

ADS functions

Functions are described below which, with the aid of the ADS interface makes some of the functions of the Windows-NT operating system (such as the output of message boxes) accessible through PLC calls.

Name	Description
ADSLOGDINT [▶ 93]	Log a DINT variable into NT Eventlog and/or Messagebox.
ADSLOGLREAL [▶ 95]	Log a (L)REAL variable into NT Eventlog and/or Messagebox.
ADSLOGSTR [▶ 96]	Log a STRING variable into NT Eventlog and/or Messagebox.
F_CreateAmsNetId [▶ 98]	Creates AmsNetId string

Name	Description
F_ScanAmsNetIds [▶ 98]	Converts AmsNetId string to array of address bytes

MEMORY functions

Number of functions which provide direct access to memory areas in the PLC runtime system.

Comment

The fact that these functions allow direct access to the physical memory means that special care is called for in applying them! Incorrect parameter values can result in a system crash, or in access to forbidden memory areas.

Name	Description
MEMCMP [▶ 99]	Compares the values of variables in two memory areas
MEMCPY [▶ 100]	Copies the values of variables from one memory area to another
MEMSET [▶ 101]	Sets the variables in a memory area to a particular value
MEMMOVE [▶ 102]	Copies the values from overlapping memory areas

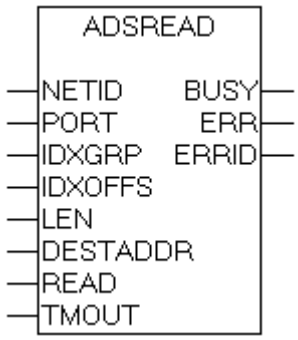
Character functions

Name	Description
F_ToASC [▶ 104]	Converts string character to ASCII number
F_ToCHR [▶ 103]	Converts ASCII number to string character

3 Function blocks

3.1 ADS Function Blocks

3.1.1 ADSREAD



This function block allows execution of an ADS read command, to request data from an ADS device.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    LEN         : UDINT;
    DESTADDR    : DWORD;
    READ        : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

LEN : Contains the number, in bytes, of the data to be read.

DESTADDR : Contains the address of the buffer which is to receive the data that has been read. The programmer is himself responsible for dimensioning the buffer to a size that can accept 'LEN' bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

READ : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

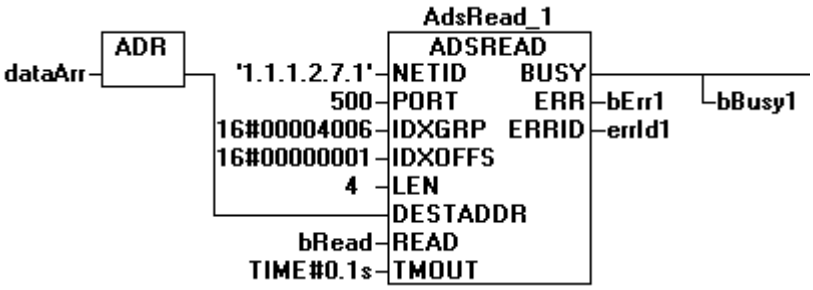
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'ErrorId'. If the block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in FBD:

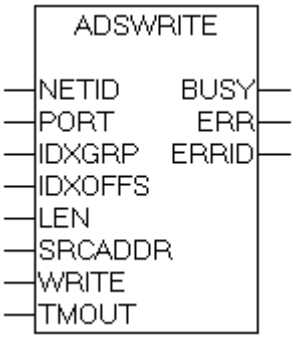


Here the error status of axis no. 6, as an element with a size of 4 bytes, is interrogated and written into the 'dataArr' buffer. The IDXGRP 00004006 (hex) and the IDXOFFS 00000001 (hex) can be found in the NC-ADS documentation.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.2 ADSWRITE



This block permits execution of an ADS write command, for the transfer of data to an ADS device.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    LEN         : UDINT;
    SRCADDR     : DWORD;
```

```
    WRITE       : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

LEN : Contains the number, in bytes, of the data to be read.

SRCADDR : Contains the address of the buffer from which the data to be written is to be fetched. The programmer is himself responsible for dimensioning the buffer to such a size that 'LEN' bytes can be taken from it. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

WRITE : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

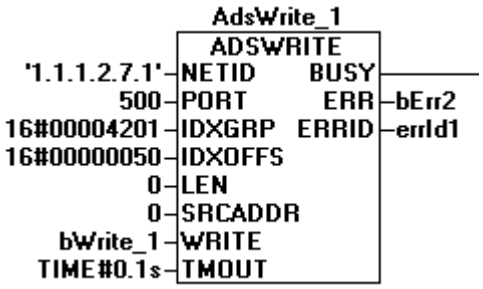
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'ErrorId'. If the block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in FBD:

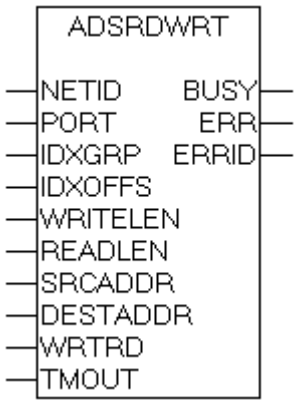


NC axis no. 1 is here deactivated through a write instruction with IDXGRP 00004201 (hex) and the IDXOFFS 00000050 (hex). To activate the axis another write instruction with the IDXOFFS 00000051 (hex) must be given. Since this write instruction does not require any parameters, the inputs LEN and SRCADDR have no significance, but must nevertheless be set to zero.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.3 ADSRDWRT



This block allows execution of a combined ADS write/read instruction. Data is transmitted to an ADS device (write) and its response data read with one call.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    WRITELEN    : UDINT;
    READLEN     : UDINT;
    SRCADDR     : DWORD;
    DESTADDR    : DWORD;
    WRTRD       : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

WRITELEN : Contains the number, in bytes, of the data that is to be written.

READLEN : Contains the number, in bytes, of the data to be read.

SRCADDR : Contains the address of the buffer from which the data to be written is to be fetched. The programmer is himself responsible for dimensioning the buffer to such a size that ‘WRITELEN’ bytes can be taken from it. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

DESTADDR : Contains the address of the buffer which is to receive the data that has been read. The programmer is himself responsible for dimensioning the buffer to a size that can accept ‘READLEN’ bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

WRTRD : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

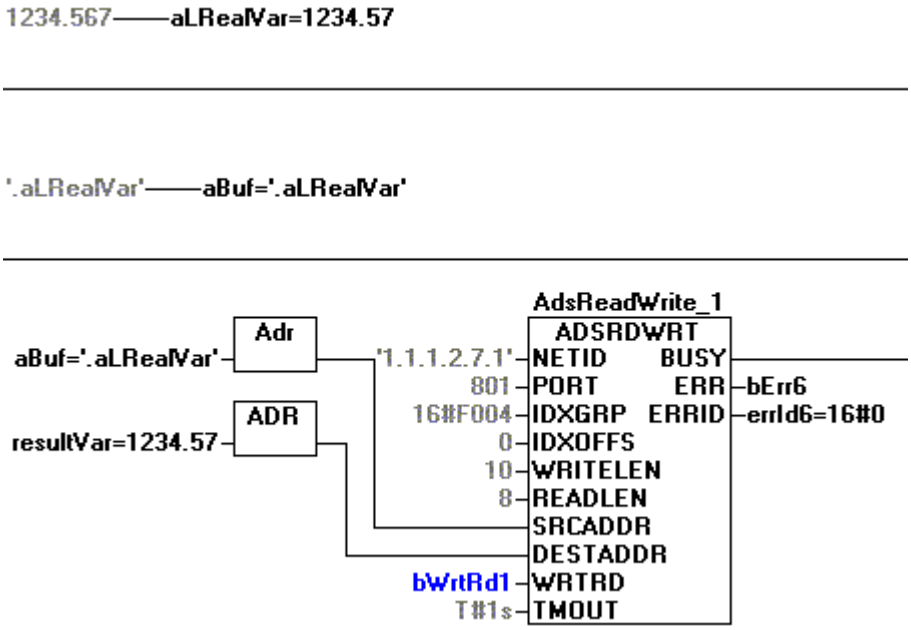
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘Timeout’ input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the block has a timeout error, ‘Error’ is TRUE and ‘ErrorId’ is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in FBD:

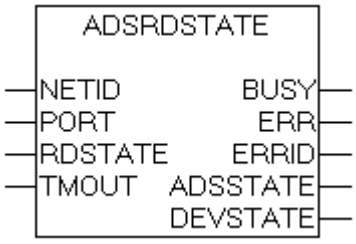


The value of the variable with the name ‘aLRealVar’ is here read from the PLC which is running on the computer with the Net-Id ‘1.1.1.2.7.1’. For this purpose, the computer address mentioned, the port number of the PLC’s first run-time system, the index group, and the index offset for reading the variable by name (F004 hex, 0) are given. The name of the variable is to be supplied to the PLC server; it is placed for this purpose in a buffer. Since the variable is global, it has a leading dot. This makes the length of the data to be written 10 characters (1 dot and 9 letters). Since the variable to be read is of type LREAL, the number of bytes to be read is 8. The address of the name buffer is given as the address for the data to be written, while for the receive data the address of an LREAL variable (‘resultVar’) is given. The diagram shows the state of the block in flow control after execution of the WriteRead instruction: the value 1234.567, which was previously contained in aLRealVar is now also contained in resultVar.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.4 ADSRDSTATE



This block permits the state of an ADS device to be requested.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    RDSTATE     : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

RDSTATE : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    ADSSTATE    : UINT;
    DEVSTATE    : UINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘Timeout’ input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the block has a timeout error, ‘Error’ is TRUE and ‘ErrorId’ is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

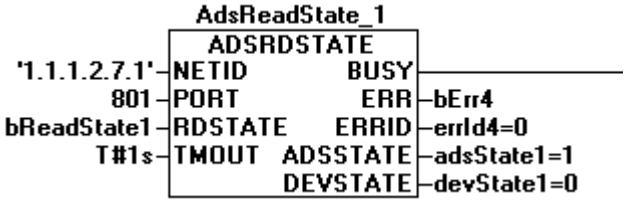
ADSSTATE : Contains the state identification code of the ADS target device. The codes returned here are specified for all ADS servers:

- ADSSTATE_INVALID =0 ;
- ADSSTATE_IDLE =1 ;

- ADSSTATE_RESET =2 ;
- ADSSTATE_INIT =3 ;
- ADSSTATE_START =4 ;
- ADSSTATE_RUN =5 ;
- ADSSTATE_STOP =6 ;
- ADSSTATE_SAVECFG =7 ;
- ADSSTATE_LOADCFG =8 ;
- ADSSTATE_POWERFAILURE =9 ;
- ADSSTATE_POWERGOOD =10 ;
- ADSSTATE_ERROR =11;

DEVSTATE : Contains the specific state identification code of the ADS target device. The codes returned here are supplementary information specific to the ADS device.

Example of calling the block in FBD:

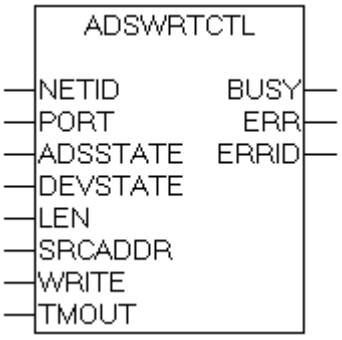


In this example the PLC run-time system 1 (port no. 801) on the computer with network address '1.1.1.2.7.1' is asked about its state. The answer is adsState = 1 (IDLE) without supplementary code devState=0.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.5 ADSWRTCTL



This block permits the execution of an ADS control command to affect the state of an ADS device, e.g. to start, stop or reset a device.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    ADSSTATE    : UINT;
    DEVSTATE    : UINT;
    LEN         : UDINT;
    SRCADDR     : DWORD;
    WRITE       : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

ADSSTATE : Contains the state identification code of the ADS target device. The codes shown here are specified for all ADS servers:

- ADSSTATE_IDLE =1 ;
- ADSSTATE_RESET =2 ;
- ADSSTATE_INIT =3 ;
- ADSSTATE_START =4 ;
- ADSSTATE_RUN=5 ;
- ADSSTATE_STOP =6 ;
- ADSSTATE_SAVECFG =7 ;
- ADSSTATE_LOADCFG =8;

DEVSTATE : Contains the specific state identification code of the ADS target device. The codes given here are supplementary information which is specific to the ADS device.

LEN : Contains the number, in bytes, of the data that is to be written.

SRCADDR : Contains the address of the buffer from which the data to be written is to be fetched. The programmer is himself responsible for dimensioning the buffer to such a size that ‘LEN’ bytes can be taken from it. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

WRITE : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

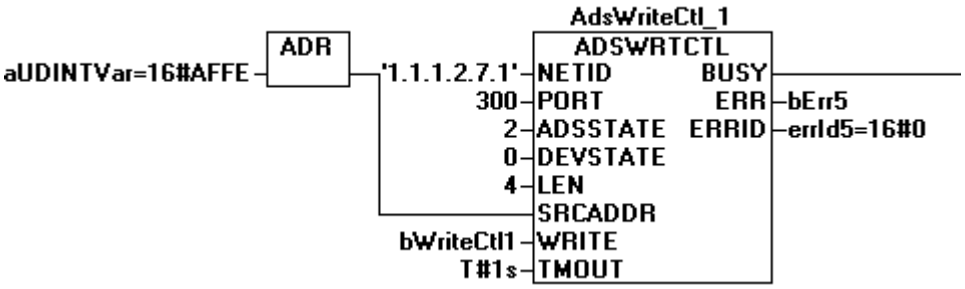
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘Timeout’ input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the block has a timeout error, ‘Error’ is TRUE and ‘ErrorId’ is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in FBD:

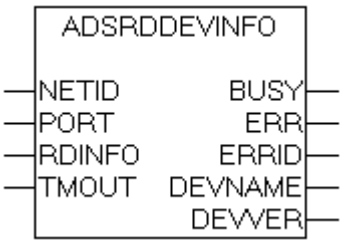


In the example a reset command (ADSSTATE=2) is sent to the I/O server (Port300), along with supplementary data hex.AFFE. As a result the I/O server executes a bus reset.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.6 ADSRDDEVINFO



The general device information can be read with this block.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    RDINFO      : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device

RDINFO : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    DEVNAME     : STRING(19);
    DEVVER      : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

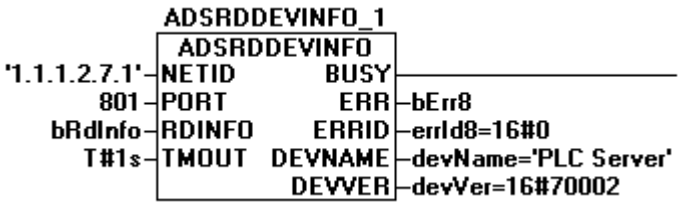
ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'ErrorId'. If the block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

DEVNAME : Contains the name of the ADS device.

DEVVER : Contains the version number of the ADS device.

Example of calling the block in FBD:

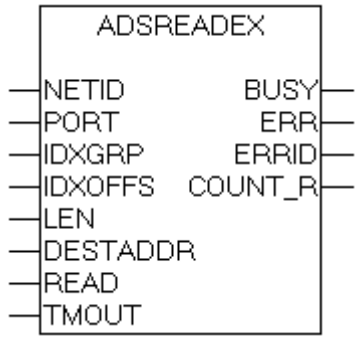


In the example, the device information of the first PLC run-time system (port 801) on computer '1.1.1.2.7.1' is read. As a result the name 'PLC Server' and the version number 02.00.7 are received.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.7 ADSREADEX



This function block allows execution of an ADS read command, to request data from an ADS device. The function block has the same functionality as the ADSREAD function block. It returns the delivery additionally also the number of actually read data bytes as parameter.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
```

```
    LEN         : UDINT;
    DESTADDR    : DWORD;
    READ        : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

LEN : Contains the number, in bytes, of the data to be read.

DESTADDR : Contains the address of the buffer which is to receive the data that has been read. The programmer is himself responsible for dimensioning the buffer to a size that can accept 'LEN' bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

READ : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    COUNT_R     : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'ErrorId'. If the block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

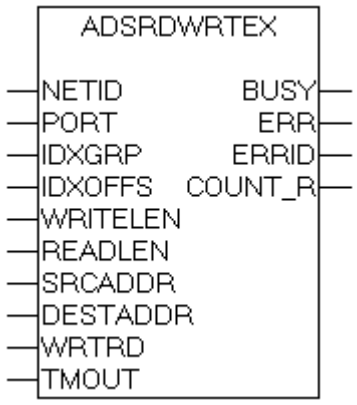
ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

COUNT_R: Number of successfully returned data bytes.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.8 ADSRDWRTEX



This block allows execution of a combined ADS write/read instruction. Data is transmitted to an ADS device (write) and its response data read with one call. Contrary to the ADSRDWRT function block ADSRDWRTEX supplys the number of actually read data bytes as parameter.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    WRITELEN    : UDINT;
    READLEN     : UDINT;
    SRCADDR     : DWORD;
    DESTADDR    : DWORD;
    WRTRD       : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

PORT : Contains the port number [▶ 108] of the ADS device.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.

WRITELEN : Contains the number, in bytes, of the data that is to be written.

READLEN : Contains the number, in bytes, of the data to be read.

SRCADDR : Contains the address of the buffer from which the data to be written is to be fetched. The programmer is himself responsible for dimensioning the buffer to such a size that ‘WRITELEN’ bytes can be taken from it. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

DESTADDR : Contains the address of the buffer which is to receive the data that has been read. The programmer is himself responsible for dimensioning the buffer to a size that can accept ‘READLEN’ bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

WRTRD : The ADS command is triggered by a rising edge at this input.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    COUNT_R     : UDINT;
END_VAR
```

BUSY : This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'ErrorId'. If the block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

COUNT_R: Number of successfully read data bytes.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2 Expanded ADS Function Blocks

The expanded ADS function blocks make it possible to create a client-server communication between an ADS device and a PLC task. With the ADS device it can be a case of a Windows application (uses the AdsDLL/AdsOcx) or another PLC run-time system. Communication between the ADS device and the PLC task is processed using the following service primitives:

- Request
- Indication
- Response
- Confirmation

Communication between an ADS device and a PLC task proceeds as follows: an ADS device sends a request to the target device (PLC task). This request is registered in the target device by an indication. The target device (PLC task) thereupon carries out a corresponding service. The service to be carried out is encoded via the index-group/offset parameter. Next the PLC sends a response to the ADS device. The response is registered as confirmation by the ADS source device.

Only one instance of the indication and response function block can meaningfully be used per PLC task (one instance of ADSREADIND, ADSREADRES, ADSWRITEIND, ADSWRITERES, ADSRDWRTIND and ADSRDWRTRES). Corresponding with the available ADS services: READ, WRITE and READ & WRITE there is an appropriate indication or response function block for each service.

Expanded ADS function blocks

Service	Name	Description
READ	ADSREADIND [▶ 27]	ADSREAD-Indication.
	ADSREADRES [▶ 31]	ADSREAD-Response
WRITE	ADSWRITEIND [▶ 28]	ADSWRITE-Indication
	ADSWRITERES [▶ 32]	ADSWRITE-Response
READ & WRITE	ADSRDWRTIND [▶ 29]	ADS-READ & WRITE-Indication
	ADSRDWRTRES [▶ 33]	ADS-READ & WRITE-Response

The ADS devices are addressed via a port address and a network address (NETID). The target address of the PLC task is formed in the following manner:

Port address of the PLC task = port number of the run-time system + task number, which should be addressed.

The current task number can be ascertained with the GETCURTASKINDEX function block.

Example:

The first PLC task of the first PLC run-time system on a local computer should be addressed. The port number of the PLC task is shown by:

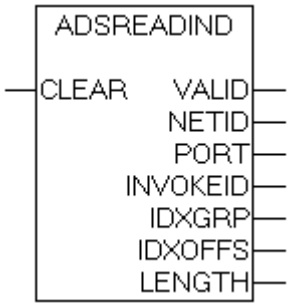
PORT = 801 + 1 = 802

NETID = ” (empty string)

Comments:

- In order for a request to be routed to the PLC task, the highest-value bit, e.g. IG:=0x80000001, must be entered in the index group parameter when the request is made.
- Each PLC task has 3 Fifos (ADSREADIND Fifo, ADSWRITEIND Fifo and ADSRDWRTIND Fifo), in which incoming requests (indications) are stored temporarily.
  Each Fifo can store up to 10 indications until they have been processed (until a response was sent). If, for example, 12 ADSREAD requests are sent to a PLC task simultaneously, 10 requests are stored in the Fifo as indications and two are acknowledged (discarded) with ADS error message 1814 (0x716). In this case, the error code should be analysed and the two failed ADSREAD requests repeated if necessary. The indications are retrieved individually from the associated Fifo by calling the ADSxxxxxxIND instance. Only then can new indications be stored successfully in the Fifo.

3.2.1 ADSREADIND



The function block registers ADSREAD-Requests at a PLC task as indications and allows them to be processed. The queuing of an indication is reported at the VALID output port by means of a rising edge. The indication is shown to have been processed by a rising edge at the CLEAR in put. A falling edge at the CLEAR input releases the function block for processing further indications. After an indication has been processed a response must be sent to the source device via the ADSREADRES [▶ 31] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter sorts the responses to the requests for the source device and is also sent back as parameter to the source device.

VAR_INPUT

```
VAR_INPUT
    CLEAR       : BOOL;
END_VAR
```

CLEAR : With a rising edge at this input an indication is reported as processed and the outputs of the ADSREADIND function block are reset. A falling edge releases the function block for the processing of further indications.

VAR_OUTPUT

```
VAR_OUTPUT
    VALID       : BOOL;
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    INVOKEID    : UDINT;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    LENGTH      : UDINT;
END_VAR
```

VALID : The output is set if an indication was registered from the function block and remains set until the latter was reported as processed by a raising edge at the CLEAR input.

NETID : Is a string, which contains the AMS network identifier [▶ 107] of the source device, from which the ADS command was sent.

PORT : Contains the port number [▶ 108] of the ADS source device, from which the ADS command was sent.

INVOKEID : Contains a handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service.

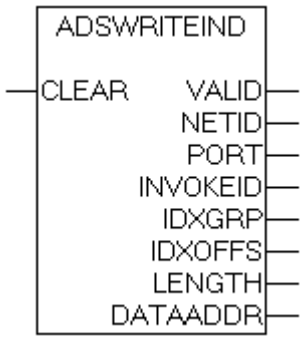
IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service.

LENGTH : Contains the number, in bytes, of the data to be read.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.2 ADSWRITEIND



The function block registers ADSWRITE-Requests to a PLC task as indications and allows them to be processed. The queuing of an indication is reported at the VALID output port by means of a rising edge. The indication is shown to have been processed by a rising edge at the CLEAR in put. A falling edge releases the function block for the processing of further indications. After an indication has been processed a response must be sent to the source device via the ADSWRITERES [▶ 32] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter sorts the responses to the requests for the source device and is also sent back as parameter to the source device.

VAR_INPUT

```
VAR_INPUT
    CLEAR       : BOOL;
END_VAR
```

CLEAR : With a rising edge at this input an indication is reported as processed and the outputs of the ADSWRITEIND function block are reset (DATAADDR = 0, LENGTH = 0 !). A falling edge releases the function block for the processing of further indications.

VAR_OUTPUT

```
VAR_OUTPUT
    VALID       : BOOL;
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    INVOKEID    : UDINT;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    LENGTH      : UDINT;
    DATAADDR    : DWORD;
END_VAR
```

VALID : The output is set if an indication was registered from the function block and remains set until the latter was reported as processed by a raising edge at the CLEAR input.

NETID : Is a string, which contains the AMS network identifier [▶ 107] of the source device, from which the ADS command was sent.

PORT : Contains the port number [▶ 108] of the ADS source device, from which the ADS command was sent.

INVOKEID : Contains a handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service.

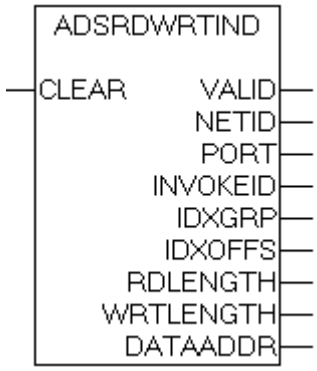
LENGTH : Contains the length, in bytes, of the data to be written.

DATAADDR : Contains the address of the data buffer, in which the data written is located.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.3 ADSRDWRTIND



The function block registers ADSRDWRT-Requests to a PLC task as indications and allows them to be processed. The queuing of an indication is reported at the VALID output port by means of a rising edge. The indication is shown to have been processed by a rising edge at the CLEAR in put. A falling edge releases the function block for the processing of further indications. After an indication has been processed a response must be sent to the source device via the ADSRDWRTRES [▶ 33] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter sorts the responses to the requests for the source device and is also sent back as parameter to the source device.

VAR_INPUT

```
VAR_INPUT
    CLEAR       : BOOL;
END_VAR
```

CLEAR : With a rising edge at this input an indication is reported as processed and the outputs of the ADSRDWRTIND function block are reset. A falling edge releases the function block for the processing of further indications.

VAR_OUTPUT

```
VAR_OUTPUT
    VALID       : BOOL;
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    INVOKEID    : UDINT;
    IDXGRP      : UDINT;
    IDXOFFS     : UDINT;
    RDLENGTH    : UDINT;
    WRTLENGTH   : UDINT;
    DATAADDR    : DWORD;
END_VAR
```

VALID : The output is set if an indication was registered from the function block and remains set until the latter was reported as processed by a raising edge at the CLEAR input.

NETID : Is a string, which contains the AMS network identifier [▶ 107] of the source device, from which the ADS command was sent.

PORT : Contains the port number [▶ 108] of the ADS source device, from which the ADS command was sent.

INVOKEID : Contains a handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.

IDXGRP : Contains the index group number (32 bit, unsigned) of the requested ADS service.

IDXOFFS : Contains the index offset number (32 bit, unsigned) of the requested ADS service.

RDLENGTH : Contains the length, in bytes, of the data to be read.

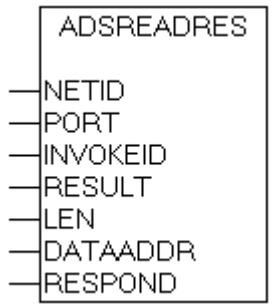
WRTLENGTH : Contains the length, in bytes, of the data to be written.

DATAADDR : Contains the address of the data buffer, in which the data written is located.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.4 ADSREADRES



The ADSREADRES function block is used to acknowledge indications of a PLC task. A response is sent to the ADS source device via a rising edge on the RESPOND input. The source device is addressed via the PORT and NETID parameters. The INVOKEID parameter sorts the responses to the requests for the source device and is adopted by the output of the ADSREADIND [▶ 27] function block. An error code can be returned to the ADS source device via the RESULT parameter.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    INVOKEID    : UDINT;
    RESULT      : UDINT;
    LEN         : UDINT;
    DATAADDR    : DWORD;
    RESPOND     : BOOL;
END_VAR
```

NETID : Is a string, which contains the AMS network identifier [▶ 107] of the source device, to which the ADS command should be sent.

PORT : Contains the port number [▶ 108] of the ADS source device, to which the response should be sent.

INVOKEID : Contains a handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.

RESULT : Contains the error code, which should be sent to the source device.

LEN : Contains the number, in bytes, of the data to be read.

DATAADDR : Contains the address of the data buffer, which should be read.

RESPOND : The function block is activated by a positive edge at this input.

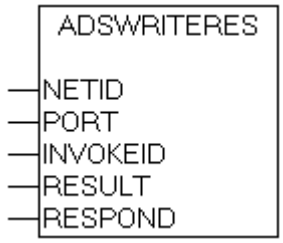
VAR_OUTPUT

```
(*none*)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.5 ADSWRITERES



The ADSWRITERES function block is used to acknowledge indications of a PLC task. A response is sent to the ADS source device via a rising edge on the RESPOND input. The source device is addressed via the PORT and NETID parameters. The INVOKEID parameter sorts the responses to the requests for the source device and is adopted by the output of the ADSWRITEIND [▶ 27] function block. An error code can be returned to the ADS source device via the RESULT parameter.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    INVOKEID    : UDINT;
    RESULT      : UDINT;
    RESPOND     : BOOL;
END_VAR
```

NETID : Is a string, which contains the AMS network identifier [▶ 107] of the source device, to which the ADS command should be sent.

PORT : Contains the port number [▶ 108] of the ADS source device, to which the ADS command should be sent.

INVOKEID : Contains a handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.

RESULT : Contains the error code, which should be sent to the source device.

RESPOND : The function block is activated by a positive edge at this input.

VAR_OUTPUT

```
(*none*)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.6 ADSRDWRTRES

The ADSRDWRTRES function block is used to acknowledge indications of a PLC task. A response is sent to the ADS source device via a rising edge on the RESPOND input. The source device is addressed via the PORT and NETID parameters. The INVOKEID parameter sorts the responses to the requests for the source device and is adopted by the output of the ADSRDWRTIND [▶ 29] function block. An error code can be returned to the ADS source device via the RESULT parameter.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : T_AmsPort;
    INVOKEID    : UDINT;
    RESULT      : UDINT;
    LEN         : UDINT;
    DATAADDR    : DWORD;
    RESPOND     : BOOL;
END_VAR
```

NETID : Is a string, which contains the AMS network identifier [▶ 107] of the source device, to which the ADS command should be sent.

PORT : Contains the port number [▶ 108] of the ADS source device, to which the ADS command should be sent.

INVOKEID : Contains a handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.

RESULT : Contains the error code, which should be sent to the source device.

LEN : Length, in bytes, of the read data.

DATAADDR : Address of the data buffer, in which the read data is located.

RESPOND : The function block is activated by a positive edge at this input.

VAR_OUTPUT

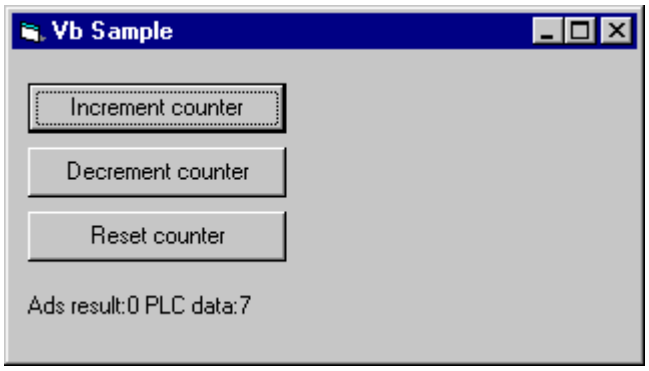
```
(*none*)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.7 Example 1: Expanded ADS function blocks

In the example application, READ-Requests are sent from a VB application to the PLC task in order to increment/decrement or reset a PLC counter variable. If successful the value of the counter variables is sent back to the VB application and outputted on the form. In order to communicate with the PLC task the VB application uses the ActiveX control: AdsOcx.



Here you can unpack the complete sources relating to the example application: https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828002187/.exe

The VB application

A connection to the PLC task is constructed in the *Form_Load*-Routine (port 802 on the local computer). The required service in the PLC task is encoded in the index group parameter:

- IG:0x80000001 -> increment the counter variable;
- IG:0x80000002 -> decrement the counter variable;
- IG:0x80000003 -> set the counter variable = 0;

So that the requests can be routed to the PLC task, the highest value bit must be set in the index group parameter. The index offset parameter is zero.

```
Option Explicit

Dim tmpData(1) As Integer
Dim AdsResult    As Integer

Private Sub Command1_Click()

    AdsResult = AdsOcx1.AdsSyncReadReq(&H80000001, &H0, 2, tmpData)
    Label1.Caption = "Ads result:" & AdsResult & " PLC data:" & tmpData(0)

End Sub

Private Sub Command2_Click()
    AdsResult = AdsOcx1.AdsSyncReadReq(&H80000002, &H0, 2, tmpData)
    Label1.Caption = "Ads result:" & AdsResult & " PLC data:" & tmpData(0)

End Sub

Private Sub Command3_Click()
    AdsResult = AdsOcx1.AdsSyncReadReq(&H80000003, &H0, 2, tmpData)
    Label1.Caption = "Ads result:" & AdsResult & " PLC data:" & tmpData(0)

End Sub

Private Sub Form_Load()
    AdsOcx1.AdsAmsServerNetId = AdsOcx1.AdsAmsClientNetId
    AdsOcx1.AdsAmsServerPort = 802 'PLC task number'
End Sub
```

The PLC program

The requests are intercepted as indications in the PLC task by an instance of the ADSREADIND [▶ 27] function block. Next the index group and index offset parameters and the required data length and validity are checked. In the CASE instruction the desired operation is implemented on the PLC variables. If successful a response is sent back by an instance of the ADSREADRES [▶ 31] function block to the caller with the current value of the PLC variables. In the case of an error an appropriate error message. At the end the CLEAR and RESPOND flags are reset in order to be able to process further indications.

```
PROGRAM MAIN
VAR
    fbReadInd   : ADSREADIND;(* Indication function block instance *)
    fbReadRes   : ADSREADRES;(* Response function block instance *)

    sNetId      : T_AmsNetID;
    nPort       : T_AmsPort;
    nInvokeId       : UDINT;
    nIdxGrp     : UDINT;
    nIdxOffs    : UDINT;
    cbLength    : UDINT;(* Requested read data/buffer byte size *)

    cbRead      : UDINT;(* Returned read data/buffer byte size *)
    pRead       : POINTER TO BYTE;(* Pointer to returned read data/buffer *)
    nErrID      : UDINT;(* Read indication result error code *)

    nCounter    : INT;(* Server data *)
END_VAR
```

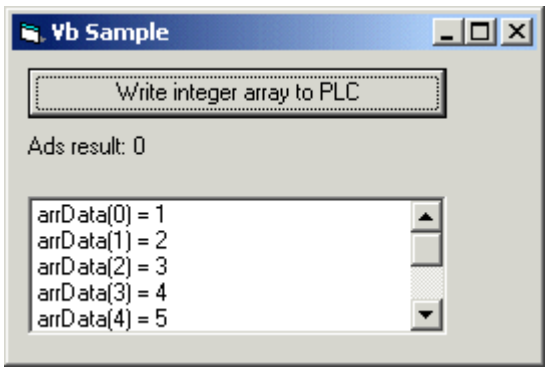
```
fbReadRes( RESPOND := FALSE );(* Reset response function block *)
fbReadInd( CLEAR := FALSE );(* Trigger indication function block *)
IF fbReadInd.VALID THEN(* Check for new indication *)

    sNetID := fbReadInd.NETID;
    nPort := fbReadInd.PORT;
    nInvokeID := fbReadInd.INVOKEID;
    nIdxGrp := fbReadInd.IDXGRP;
    nIdxOffs := fbReadInd.IDXOFFS;
    cbLength := fbReadInd.LENGTH;

    cbRead := 0;
    pRead := 0;
    nErrID  := 16#701;(* ADS error: Service not supported by server *)


    CASE nIdxGrp OF
        (*-----------------------------------------------------------------------------------------*)
        16#80000001:
            CASE nIdxOffs OF
                0:(* Increment counter value *)
                    IF cbLength >= SIZEOF(nCounter) THEN
                        nCounter := nCounter + 1;
                        cbRead := SIZEOF(nCounter);
                        pRead := ADR(nCounter);
                        nErrID := 0;
                    ELSE (* ADS error (example): Invalid size *)
                        nErrID := 16#705;
                    END_IF
            ELSE (* ADS error (example): Invalid index offset *)
                nErrID := 16#703;
            END_CASE
        (*-----------------------------------------------------------------------------------------*)
        16#80000002:
            CASE nIdxOffs OF
                0:(* Decrement counter value *)
                    IF cbLength >= SIZEOF(nCounter) THEN
                        nCounter := nCounter - 1;
                        cbRead := SIZEOF(nCounter);
                        pRead := ADR(nCounter);
                        nErrID := 0;
                    ELSE(* ADS error (example): Invalid size *)
                        nErrID := 16#705;
                    END_IF
            ELSE (* ADS error (example): Invalid index offset *)
                nErrID := 16#703;
            END_CASE
        (*-----------------------------------------------------------------------------------------*)
```

```
        16#80000003:
            CASE nIdxOffs OF
                0:(* Reset counter value *)
                    IF cbLength >= SIZEOF(nCounter) THEN
                        nCounter := 0;
                        cbRead := SIZEOF(nCounter);
                        pRead := ADR(nCounter);
                        nErrID := 0;
                    ELSE(* ADS error (example): Invalid size *)
                        nErrID := 16#705;
                    END_IF
            ELSE (* ADS error (example): Service is not supported by server *)
                nErrID := 16#701;(* ADS error: Service not supported *)
            END_CASE

    ELSE (* ADS error (example): Invalid index group *)
        nErrID := 16#702;
    END_CASE

    fbReadRes(      NETID := sNetID,
                PORT := nPort,
                INVOKEID := nInvokeID,
                LEN := cbRead,
                DATAADDR := pRead,
                RESULT := nErrID,
                RESPOND := TRUE );(* Send read response *)

    fbReadInd( CLEAR := TRUE ); (* Clear indication entry *)
END_IF
```

Here you can unpack the complete sources relating to the example application: https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828002187/.exe

3.2.8 Example 2: Expanded ADS function blocks

The VB application sends an array containing integer values to a PLC task. The most recently sent values are added to a list, and are copied into a corresponding array variable in the PLC. The VB application uses a WRITE request to send the data. The ADSWRITEIND function block is used in the PLC in order to receive the data, while the ADSWRITERES function block is used to acknowledge the WRITE request. In order to communicate with the PLC task the VB application uses the ActiveX control: AdsOcx.



Here you can unpack the complete sources relating to the example application: https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828002187/.exe

The VB application

A connection to the PLC task is constructed in the *Form_Load*-Routine (port 802 on the local computer). The desired service from the PLC task is encoded in the index group and index offset parameters. E.g.:

- IG:0x80000005 and
- IO:0x00000007-> Copy the sent data into the array in the PLC.

So that the requests can be routed to the PLC task, the highest value bit must be set in the index group parameter.

```
Option Explicit

Dim AdsResult     As Integer
Dim arrData(0 To 9)   As Integer

Private Sub cmdWrite_Click()
    Call List1.Clear

    Dim i As Long
    For i = LBound(arrData) To UBound(arrData)
    arrData(i) = arrData(i) + 1 'change values
    Call List1.AddItem("arrData(" & i & ") = " & arrData(i))
    Next i

    'calculate the byte length parameter
    Dim cbWriteSize As Long
    cbWriteSize = (UBound(arrData) - LBound(arrData) + 1) * LenB(arrData(LBound(arrData)))

    AdsResult = AdsOcx1.AdsSyncWriteReq(&H80000005, &H7, cbWriteSize, arrData) 'send data to PLC
    Label1.Caption = "Ads result: " & AdsResult

End Sub

Private Sub Form_Load()
    AdsOcx1.AdsAmsServerNetId = AdsOcx1.AdsAmsClientNetId
    AdsOcx1.AdsAmsServerPort = 802 'PLC task number'

    Dim i As Long
    For i = LBound(arrData) To UBound(arrData)
    arrData(i) = i    'init data
    Next i
End Sub
```

The PLC program

The requests are intercepted as indications in the PLC task by an instance of the ADSWRITEIND [▶ 28] function block. Following this, the index group, index offset and transmitted data length parameters are checked for validity, and the desired operation is carried out on the PLC variable. The next step is for a response to be returned to the caller (including an error code, if appropriate) by an instance of the ADSWRITERES [▶ 32] function block. At the end the CLEAR and RESPOND flags are reset in order to be able to process further indications.

With the rising edge at the CLEAR input of the ADSWRITEIND function block the address pointer to the most recently sent data becomes invalid (== NULL). For this reason the sent data is first copied into the PLC variable before the CLEAR input is set to TRUE.

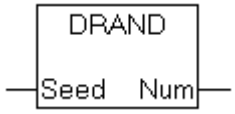
```
PROGRAM MAIN
VAR
    fbWriteInd      : ADSWRITEIND;
    fbWriteRes      : ADSWRITERES;

    sNetId      : T_AmsNetID;
    nPort       : T_AmsPort;
    nInvokeId       : UDINT;
    nIdxGrp     : UDINT;
    nIdxOffs        : UDINT;
    cbWrite     : UDINT;(* Byte size of written data *)
    pWrite      : POINTER TO BYTE;(* Pointer to written data buffer *)
    nResult     : UDINT;(* Write indication result error code *)

    arrInt      : ARRAY[0..9] OF INT;(* Server data *)
END_VAR
```

```
fbWriteRes( RESPOND := FALSE );(* Reset response function block *)
fbWriteInd( CLEAR := FALSE );(* Trigger indication function block *)
IF ( fbWriteInd.VALID ) THEN

    sNetId      := fbWriteInd.NETID;
    nPort       := fbWriteInd.PORT;
    nInvokeId       := fbWriteInd.INVOKEID;
    nIdxGrp         := fbWriteInd.IDXGRP;
    nIdxOffs        := fbWriteInd.IDXOFFS;

    cbWrite     := fbWriteInd.LENGTH;
    pWrite      := fbWriteInd.DATAADDR;
    nResult     := 16#701;(* ADS error: Service not supported by server *)

    CASE nIdxGrp OF
        16#80000005:
            CASE nIdxOffs OF
                16#00000007:
                    IF  cbWrite <= SIZEOF( arrInt ) THEN
                        MEMCPY( ADR( arrInt ), pWrite, MIN( cbWrite, SIZEOF(arrInt) ) );
                        nResult := 0;
                    ELSE(* ADS error (example): Invalid size *)
                        nResult := 16#705;
                    END_IF
            ELSE(* ADS error (example): Invalid index offset *)
                nResult := 16#703;
            END_CASE
    ELSE(* ADS error (example): Invalid index group *)
        nResult := 16#702;
    END_CASE

    fbWriteRes( NETID := sNetId,
                PORT := nPort,
                INVOKEID := nInvokeId,
                RESULT := nResult,
                RESPOND :=  TRUE ); (* Send write response *)

    fbWriteInd( CLEAR := TRUE ); (* Clear indication entry *)
END_IF
```

Here you can unpack the complete sources relating to the example application: https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828002187/.exe

3.3 General Function Blocks

3.3.1 DRAND



Instances of function blocks are created according to IEC61131-3, and then called, or otherwise accessed, from with the PLC program using the instance names.
The function block permits generation of a (pseudo-) random number of type LREAL.

VAR_INPUT

```
VAR_INPUT
    Seed    : INT;
END_VAR
```

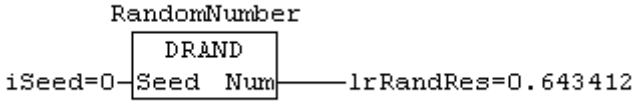
Seed : Initial value for specification of the random number series.

VAR_OUTPUT

```
VAR_OUTPUT
    Num     : LREAL;
END_VAR
```

Num : This output returns a pseudo-random number in the range 0.0 .. 1.0 with double accuracy. The generator here creates a number series with 1075 stochastic values per period.

Example of calling the block in FBD:

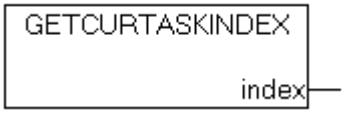


In the example the LREAL value 0.643412 is generated and returned. The input parameter ”Seed" affects the initial value of the series. If, for instance, a deterministically reproducible random number series is desired in different sessions, and identical ”Seed" value must be used.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.3.2 GETCURTASKINDEX



This function block finds the task index of the task from which it is called.

VAR_INPUT

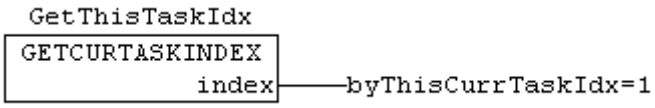
```
(*none*)
```

VAR_OUTPUT

```
VAR_OUTPUT
    index   : BYTE;
END_VAR
```

index : Returns the current task index of the calling task (1..4).

Example of calling the block in FBD:

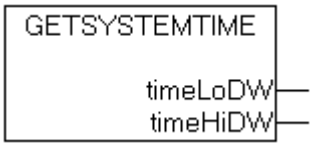


Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.4 Time Function Blocks

3.4.1 GETSYSTEMTIME



With this block the operating system time stamp can be read. The time stamp is a 64 bit integer value, with a precision of 100ns, which is updated with every call of the PLC. Amongst other uses, it can be utilised for timing tasks or time measurements. One unit is equivalent to 100 ns. The reason for which this service is implemented as a block and not as a function is simply in the fact that two values must be returned, which, by definition, cannot be done by a function.

VAR_INPUT

```
(*none*)
```

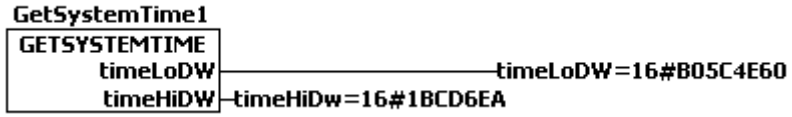
VAR_OUTPUT

```
VAR_OUTPUT
    timeLoDW        : UDINT;
    timeHiDW        : UDINT;
END_VAR
```

timeLoDW : Contains the low-value 4 bytes of the time stamp.

timeHiDW : Contains the high-value 4 bytes of the time stamp.

Example of calling the block in FBD:

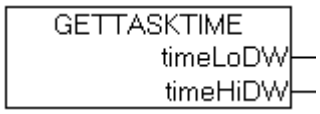


The example illustrates calling the block via the instance ‘GetSystemTime1’, and delivers the 64 bit, integer value (hex) 1BCD6EAB05C4E60 as the time stamp.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.4.2 GETTASKTIME



This function supplies the task target start time. The returned time is the target time of the task where the function block instance is called. The time stamp is a 64 bit integer value, with a precision of 100ns. Amongst other uses, it can be utilised for timing tasks or time measurements. One unit is equivalent to 100ns and epresents number of 100ns since 1 January 1601.

VAR_INPUT

```
(*none*)
```

VAR_OUTPUT

```
VAR_OUTPUT
    timeLoDW        : UDINT;
    timeHiDW        : UDINT;
END_VAR
```

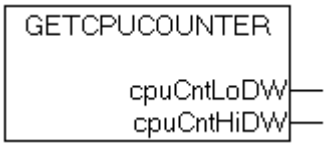
timeLoDW : Contains the low-value 4 bytes of the time stamp.

timeHiDW : Contains the high-value 4 bytes of the time stamp.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcSystem.Lib

3.4.3 GETCPUCOUNTER



The CPU cycle counter can be read with this function block. The numerical value is a relative 64 bit integer, which, independently of the CPU's internal clock rate, is output in a form converted into 100ns ticks. The number is refreshed to a precision of 100ns with every call by the PLC system, and can be used, for instance, for timing tasks. One unit is equivalent to 100 ns. The reason for which this service is implemented as a block and not as a function is simply in the fact that two values must be returned, which, by definition, cannot be done by a function.

VAR_INPUT

```
(*none*)
```

VAR_OUTPUT

```
VAR_OUTPUT
    cpuCntLoDW      : UDINT;
    cpuCntHiDW      : UDINT;
END_VAR
```

cpuCntLoDW : Contains the low-value 4 bytes of the numerical value.

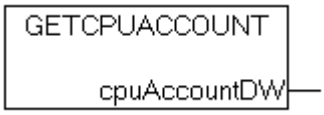
cpuCntHiDW : Contains the high-value 4 bytes of the numerical value.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.4.4 GETCPUACCOUNT

This functionality is not available on PLC runtime system under Windows CE!



The PLC task cycle ticker can be read with this function block. The PLC task cycle ticker is only incremented while the task is being executed. The numerical value is a 32 bit integer, which, independently of the CPU's internal clock rate, is output in a form converted into 100ns ticks. The number is refreshed to a precision of 100ns every time the PLC task is called, and can be used, for instance, for timing purposes. One unit is equivalent to 100 ns.

VAR_INPUT

```
(*none*)
```

VAR_OUTPUT

```
VAR_OUTPUT
    cpuAccountDW    :UDINT;
END_VAR
```

cpuAccountDW: the current value of the PLC task ticker;

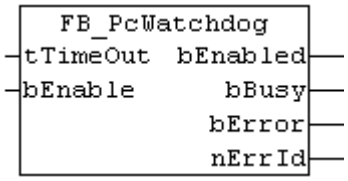
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib

3.5 Watchdog Function Blocks

3.5.1 FB_PcWatchdog

This functionality is only available on IPCs with the following mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051.



The function block FB_PcWatchdog enables a hardware watchdog on the PC. The watchdog is enabled via bEnable = TRUE and the timeout period. The timeout time can range between 1 and 255 seconds. The watchdog is enabled via bEnable = TRUE and tTimeOut >= 1 s.

Once the watchdog has been activated, the function block must be called cyclically at shorter intervals than tTimeOut, since the PC restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be disabled via bEnable = FALSE or tTimeOut = 0.

The watchdog must be disabled before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is enabled, because otherwise the PC would reboot immediately once the timeout has elapsed.

VAR_INPUT

```
VAR_INPUT
    tTimeOut        : TIME;
    bEnable     : BOOL;
END_VAR
```

tTimeOut: watchdog time, after which a restart is performed.

bEnable: error when enabling or disabling the watchdog.

VAR_OUTPUT

```
VAR_OUTPUT
    bEnabled        : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bEnabled : TRUE enables the K-bus Watchdog, FALSE disables the K-bus Watchdog.

bBusy : this output remains TRUE until the function block has executed a command.

bError: this output is set to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Sample of calling the function block in ST:

```
VAR

    fbPcWatchDog  : FB_PcWatchdog;

    tWDTime       :
TIME := T#10s;

    bEnableWD     : BOOL;

    bWDActive     : BOOL;

END_VAR
```

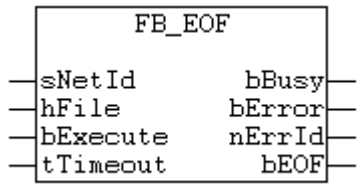
```
IF bEnableWD OR bWDActive THEN

    fbPcWatchDog(tTimeOut := tWDTime, bEnable :=
bEnableWD);

    bWDActive := fbPcWatchDog.bEnabled;

END_IF
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.9.0 from Build 1004	PC with mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051	TcSystem.Lib

3.6 File Function Blocks

3.6.1 FB_EOF



The function block "FB_EOF" tests for the end-of-file.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    bEOF        : BOOL;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘tTimeout’ input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘nErrId’.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

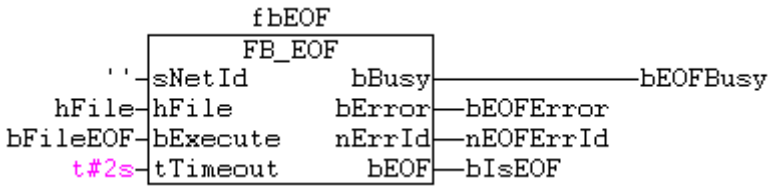
bEOF: This output is switched to TRUE if the end of file is reached.

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

```
PROGRAM Test
VAR
    fbEOF       : FB_EOF;
    hFile       : UINT;
```

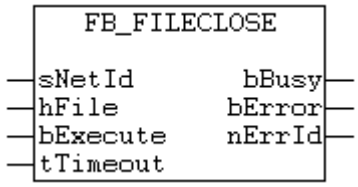
```
    bFileEOF    : BOOL;
    bEOFBusy    : BOOL;
    bEOFError   : BOOL;
    nEOFErrorId : UDINT;
    bIsEOF      : BOOL;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.2 FB_FileClose



With this function block the file can be closed, and thereby placed into a defined state for further processing by other programs.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle created with FB_FileOpen.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

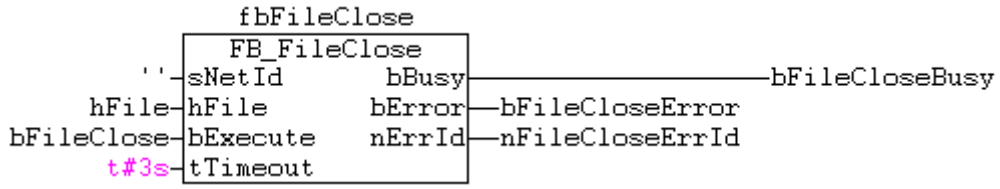
bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

```
PROGRAM Test
VAR
    fbFileClose      : FB_FileClose;
    hFile            : UINT;
    bFileClose       : BOOL;
    bFileCloseBusy   : BOOL;
    bFileCloseError  : BOOL;
    nFileCloseErrorId: UDINT;
END_VAR
```

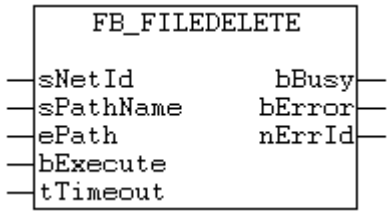


Here the file associated with the file handle (which was itself generated by ”FB_FileOpen") is closed again.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.3 FB_FileDelete



With this function block the existing file can be deleted.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    sPathName   : T_MaxString;            (* file path and name *)
    ePath       : E_OpenPath := PATH_GENERIC;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

sPathName : Contains the path and filename [▶ 109] for the file to be deleted.

ePath : The variable of this type selects generic or one of the TwinCAT system paths [▶ 110] on the target device to perform the file open operation.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

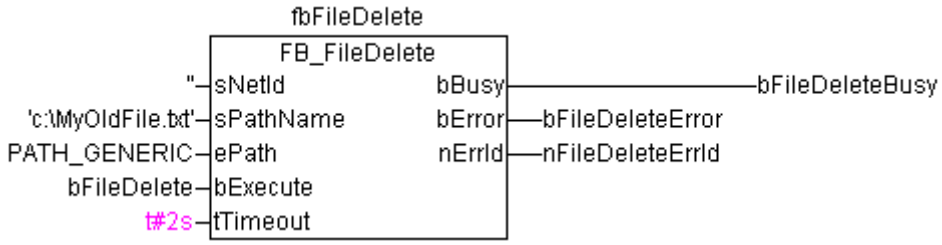
bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x70C	File not found. Invalid sPathName or ePath parameter.

Example of calling the block in FBD:

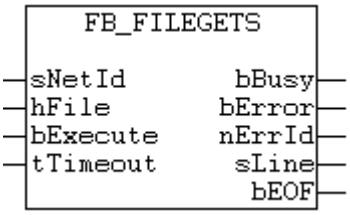
```
PROGRAM Test
VAR
    fbFileDelete    : FB_FileDelete;
    bFileDelete     : BOOL;
    bFileDeleteBusy : BOOL;
    bFileDeleteError: BOOL;
    nFileDeleteErrId: UDINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.4 FB_FileGets



The function block "FB_FileGets" reads zero terminated strings from a file. The string is read up to the line feed character and inclusive the line feed character, or to the end of the file, or until the number of characters read is equal to max.length of *sLine*. The result stored in *sLine* is appended with a null character. The line feed character, if read, is included in the string. The file must previously have been opened in the text mode.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    sLine       : T_MaxString;
    bEOF        : BOOL;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘tTimeout’ input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘nErrId’.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

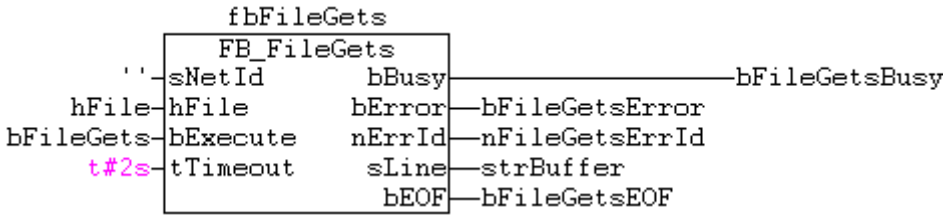
sLine : Contains the read string [▶ 109].

bEOF : This output is switched to TRUE if the end of file is reached and no more data bytes could be read (cbRead=0). This output is not set if some data bytes could be read (cbRead>0).

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70A	No memory for read buffer.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

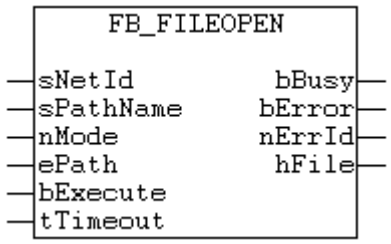
```
PROGRAM Test
VAR
    fbFileGets      : FB_FileGets;
    hFile           : UINT;
    bFileGets       : BOOL;
    bFileGetsBusy   : BOOL;
    bFileGetsError  : BOOL;
    nFileGetsErrorId: UDINT;
    strBuffer       : STRING;
    bFileGetsEOF    :BOOL;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.5 FB_FileOpen



With this function block a new file can be created, or a closed existing file can be re-opened for further processing.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    sPathName   : T_MaxString;
    nMode       : DWORD;
    ePath       : E_OpenPath := PATH_GENERIC;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

sPathName : contains the path and file name [▶ 109] of the file to be opened.

The path can only point to the local computer's file system. This means that network paths cannot be used here!

nMode : contains the mode in which the file is to be opened. The codes listed below are the various opening modes which are already pre-defined as constants in the library and which can accordingly be passed symbolically to the function block. The opening modes can be ORed. The opening modes can be combined, similar to the opening modes of the *fopen* function in C or C++.

Modes	Description
FOPEN_MODEREAD	"r": opens a file for reading. An error is returned if the file cannot be found or does not exist.
FOPEN_MODEWRITE	"w": opens an empty file for writing. If the file already exists, it is overwritten.
FOPEN_MODEAPPEND	"a": opens a file for writing at the end of the file (append). If the file does not exist, a new file is created.
FOPEN_MODEREAD OR FOPEN_MODEPLUS	"r+": opens a file for reading and writing. The file must exist.
FOPEN_MODEWRITE OR FOPEN_MODEPLUS	"w+": opens an empty file for reading and writing. If the file already exists, it is overwritten.
FOPEN_MODEAPPEND OR FOPEN_MODEPLUS	"a+": opens a file for reading and writing at the end of the file (append). If the file does not exist, a new file is created. For this, the memory path must be known, otherwise error 1804 appears. All write operations are always performed at end of a file, if the file was opened in the modes "a" or "a+". The file pointer can be moved with FB_FileSeek, although for writing it is moved to the end of the file by default, i.e. existing data cannot be overwritten.
FOPEN_MODEBINARY	"b": opens the file in binary mode
FOPEN_MODETEXT	"t": opens the file in text mode

ePath : this input can be used to select a TwinCAT system path [▶ 110] on the target device for opening the file.

bExecute : the function block is enabled by a rising edge at this input.

tTimeout : states the timeout period that must not be exceeded when executing the ADS command.

Function specific ADS error code	Possible cause
0x703	Unknown or invalid nMode or ePath parameter.
0x70C	File not found. Invalid file name or file path.
0x716	No further free file handles.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    hFile       : UINT;             (* file handle *)
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘tTimeout’ input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘nErrId’.

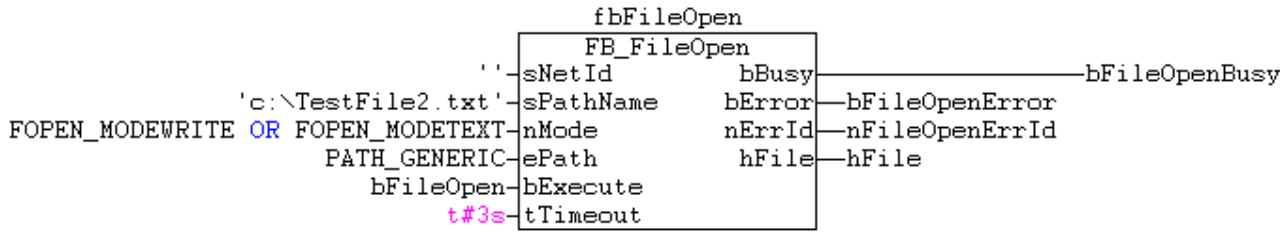
nErrId: Contains the command-specific ADS error code of the most recently executed command.

hFile: Contains the file handle created for the file when opening has been successful.

Function specific ADS error code	Possible reason
0x703	Unknown or invalid nMode or ePath parameter.
0x70C	File not found. Invalid file name or file path.
0x716	No more free file handles.

Example of calling the block in FBD:

```
PROGRAM Test
VAR
    fbFileOpen      : FB_FileOpen;
    bFileOpen       : BOOL;
    bFileOpenBusy   : BOOL;
    bFileOpenError  : BOOL;
    nFileOpenErrId  : UDINT;
    hFile           : UINT;
END_VAR
```



This should create (or overwrite) the file ”TestFile2.txt" in the root directory of drive ”C:" in the text mode.

For the opening mode a maximum of 3 parameters may be ORed**:**
Mode = [Parameter1] OR [Parameter2] OR [Paramerter3]

Parameter1 may have only one subordinate value:

- FOPEN_MODEREAD
- FOPEN_MODEWRITE
- FOPEN_MODEAPPEND

Parameter2 may have only one subordinate value:

- FOPEN_MODEPLUS

Parameter3 may have only one subordinate value:

- FOPEN_MODEBINARY
- FOPEN_MODETEXT

If no binary or text mode is specified, the file opens in a mode defined by an operating system variable. In most cases, the file will then open in text mode. However, it is not possible to make a clear statement. It is useful to always specify the text or binary mode. This system variable cannot be changed in the PLC!
This results in the following permissible combinations:

Text file opening modes	Binary file opening modes
FOPEN_MODEREAD OR FOPEN_MODETEXT	FOPEN_MODEREAD OR FOPEN_MODEBINARY
FOPEN_MODEWRITE OR FOPEN_MODETEXT	FOPEN_MODEWRITE OR FOPEN_MODEBINARY
FOPEN_MODEAPPEND OR FOPEN_MODETEXT	FOPEN_MODEAPPEND OR FOPEN_MODEBINARY
FOPEN_MODEREAD OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEREAD OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY
FOPEN_MODEWRITE OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEWRITE OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY

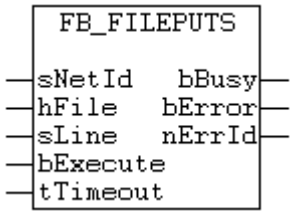
Text file opening modes	Binary file opening modes
FOPEN_MODEAPPEND OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEAPPEND OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY

All other combinations are wrong. Examples of invalid opening modes:
FOPEN_MODEBINARY OR FOPEN_MODETEXT
FOPEN_MODEWRITE OR FOPEN_MODEAPPEND

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.6 FB_FilePuts



The function block "FB_FilePuts" writes strings to the file. The string's terminating null character ('$00') will not be written. The file must previously have been opened in the text mode.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    sLine       : T_MaxString;    (* string to write *)
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen.

sLine : Contains the PLC string [▶ 109] to write.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

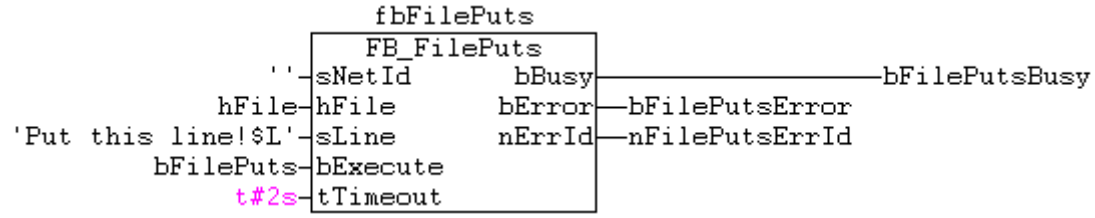
bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘nErrId’.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

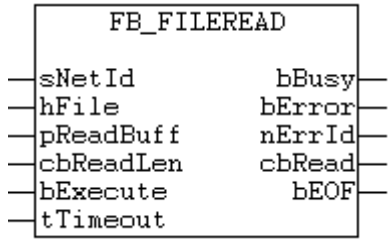
```
PROGRAM Test
VAR
    fbFilePuts      : FB_FilePuts;
    hFile           : UINT;
    bFilePuts       : BOOL;
    bFilePutsBusy   : BOOL;
    bFilePutsError  : BOOL;
    nFilePutsErrorId: UDINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.7 FB_FileRead



With this function block the contents of an already opened file can be read. The file must have been already opened for read access.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    pReadBuff   : DWORD;
    cbReadLen   : UDINT;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId :Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen.

pReadBuff : Contains the address of the buffer for the read data. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

cbReadLen : Contains the number of bytes to be read.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    cbRead      : UDINT;
    bEOF        : BOOL;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

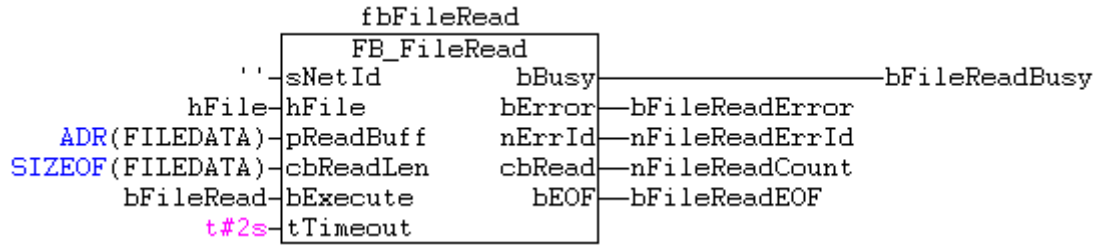
cbRead : Contains the number of bytes currently read.

bEOF : This output is switched to TRUE if an end of file is reached and no more data bytes could be read (cbRead=0). This output is not set if some data bytes could be read (cbRead>0).

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70A	No memory for read buffer.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

```
PROGRAM Test
VAR
    fbFileRead      : FB_FileRead;
    hFile           : UINT;
    bFileRead       : BOOL;
    bFileReadBusy   : BOOL;
    bFileReadError  : BOOL;
    nFileReadErrorId: UDINT;
    nFileReadCount  : UDINT;
    bFileReadEOF    : BOOL;
    FILEDATA        : ARRAY[0..9] OF BYTE;
END_VAR
```

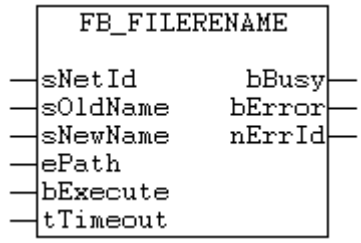


After a rising edge at ”bFileRead" and successful execution of the read instruction the currently read bytes from the file are found in ”FILEDATA". The number of bytes actually read in the previous read procedure can be determined from the parameter ”cbRead".

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.8 FB_FileRename



With this function block the existing file can be renamed.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    sOldName    : T_MaxString;
    sNewName    : T_MaxString;
    ePath       : E_OpenPath := PATH_GENERIC;      (* Default: generic file path*)
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

sOldName : Contains the old path and filename [▶ 109] for the file to be renamed.

sNewName : Contains the new path and filename for the file to be renamed.

ePath : The variable of this type selects generic or one of the TwinCAT system paths [▶ 110] on the target device to perform the file open operation.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

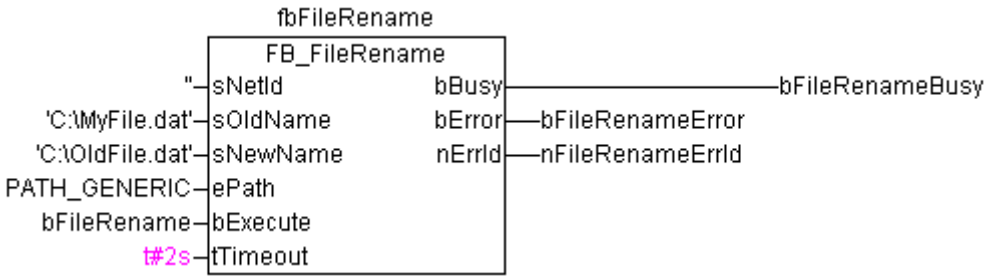
bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x70C	File not found. Invalid sOldName, sNewName or ePath parameter.

Example of calling the block in FBD:

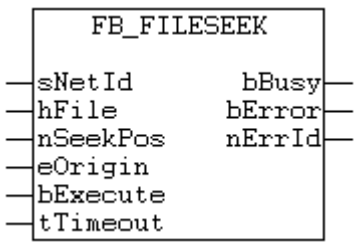
```
PROGRAM Test
VAR
    fbFileRename    : FB_FileRename;
    bFileRename     : BOOL;
    bFileRenameBusy : BOOL;
    bFileRenameError: BOOL;
    nFileRenameErrId: UDINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.9 FB_FileSeek



With this function block the file pointer of an opened file can be set to a definable position.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    nSeekPos    : DINT;          (* new seek pointer position *)
    eOrigin     : E_SeekOrigin:= SEEK_SET;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen

nSeekPos : Contains the desired (new) target position of the file pointer.

eOrigin : Contains the relative position [▶ 110] for the move.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

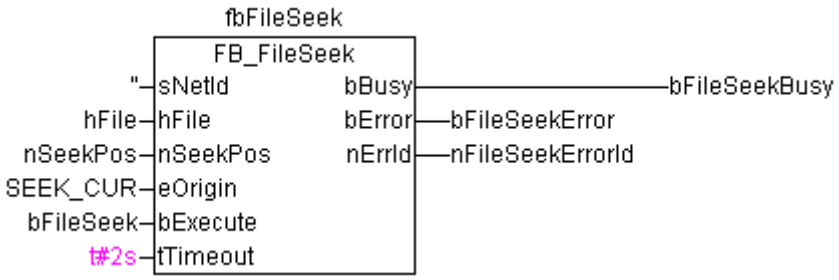
bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

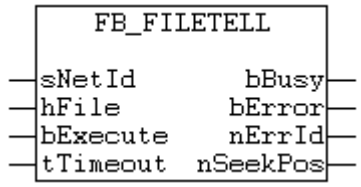
```
PROGRAM Test
VAR
    fbFileSeek      : FB_FileSeek;
    hFile           : UINT;
    nSeekPos        : DINT;
    bFileSeek       : BOOL;
    bFileSeekBusy   : BOOL;
    bFileSeekError  : BOOL;
    nFileSeekErrorId: UDINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.10 FB_FileTell



The function block "Fb_FileTell" gets the current position of the file pointer. The position is expressed as an offset relative to the beginning of the stream.

Note that when a file is opened for appending data, the current file position is determined by the last I/O operation, not by where the next write would occur. For example, if a file is opened for an append and the last operation was a read, the file position is the point where the next read operation would start, not where the next write would start. (When a file is opened for appending, the file position is moved to end of file before any write operation.) If no I/O operation has yet occurred on a file opened for appending, the file position is the beginning of the file.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    hFile       : UINT;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    nSeekPos    : DINT;         (*      On error, nSEEKPOS returns -1 *)
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

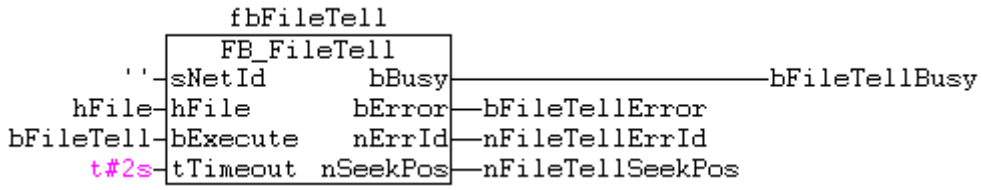
nErrId: Contains the command-specific ADS error code of the most recently executed command.

nSeekPos : This output returns the actual position of the file pointer.

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

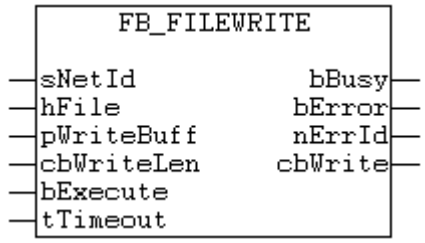
```
PROGRAM Test
VAR
    fbFileTell      : FB_FileTell;
    hFile           : UINT;
    bFileTell       : BOOL;
    bFileTellBusy   : BOOL;
    bFileTellError  : BOOL;
    nFileTellErrorId: UDINT;
    nFileTellSeekPos: DINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.11 FB_FileWrite



Data can be written into a file with this function block. The file must previously have been created and opened in the write mode.

VAR_INPUT

```
VAR_INPUT
    sNetId          : T_AmsNetId;
    hFile           : UINT;
    pWriteBuff      : DWORD;
    cbWriteLen      : UDINT;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

hFile : Is occupied by the file handle already created by FB_FileOpen.

pWriteBuff : Contains the address of the buffer containing the data to be written. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.

cbWriteLen : Contains the number of bytes to be written.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    cbWrite     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the ‘tTimeout’ input. While bBusy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘nErrId’.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

cbWrite : Contains the number of bytes successfully written.

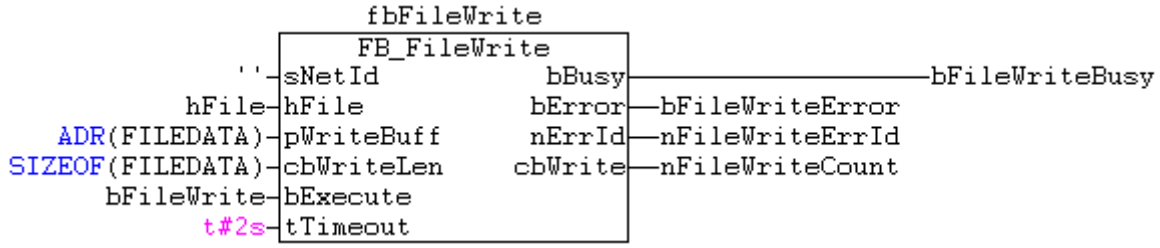
If a write error occurs the number of successful written data bytes is smaller than the requested length (*cbWriteLen*) or zero. A write error occurs e.g. if the data medium is full. The outputs *bError* and *nErrID* are not set if write error occur. Because the PLC application knows the number of data bytes to be written, the real written length is compared with the requested length and write errors can be found. If a write error occurs the internal data pointer has an undefined position.

Function specific ADS error code	Possible reason
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with 'obsolete' FILEOPEN function block).

Example of calling the block in FBD:

```
PROGRAM Test
VAR
    fbFileWrite      : FB_FileWrite;
    hFile            : UINT;
    bFileWrite       : BOOL;
    bFileWriteBusy   : BOOL;
    bFileWriteError  : BOOL;
    nFileWriteErrorId: UDINT;
```

```
    nFileWriteCount  : UDINT;
    FILEDATA         :ARRAY[0..9] OF BYTE;
END_VAR
```



In the example, after a rising edge at ”bFileWrite", 10 bytes of the array ”FILEDATA " are written to the file.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.12 Example: File access from the PLC

System requirements:

- TwinCAT 2.9;

This example illustrates use of the PLC function blocks from the TcSystem.Lib for file access. A new function block, FB_FileCopy, is implemented with the aid of the existing function blocks. This block can be used, for instance, to copy binary data on the local TwinCAT PC or from a remote TwinCAT PC to the local TwinCAT PC. The function block cannot be used to access network drives. A rising edge at the bExecute input of the FB_FileCopy block results in execution of the following steps.

a) Open the source and destination files;

b) Read the source file into a buffer;

c) Write the bytes that have been read from the buffer into the destination file;

d) Check whether the end of the source file has been reached. If not, then repeat b) and c). If yes, then jump to e);

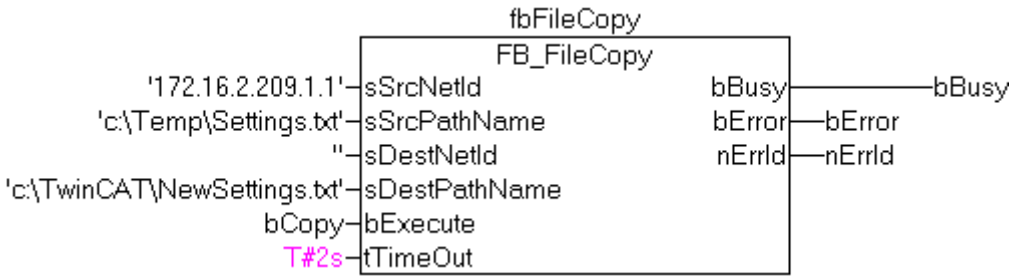
e) Close the source and destination files;

The file is copied one segment at a time. In this example, the size of the buffer has been specified as 1000 bytes, but this can be modified. The complete source code for the example project can be unpacked from here: https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828003595/.exe.

The PLC program:

```
PROGRAM MAIN
VAR
    fbFileCopy  : FB_FileCopy;
    bCopy       : BOOL;
```

```
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```



In this example, the file *Settings.txt* is copied from the *Temp* directory of a remote TwinCAT PC having network address "172.16.2.209.1.1" to the *TwinCAT* directory on the local TwinCAT PC.

The FB_FileCopy Function Block

Interface:

```
FUNCTION_BLOCK FB_FileCopy
VAR_INPUT
    sSrcNetId           : T_AmsNetId;(* TwinCAT network address of the source file *)
    sSrcPathName        : T_MaxString;(* Source file path and name *)
    sDestNetId          : T_AmsNetId;(* TwinCAT network address of the destination file *)
    sDestPathName       : T_MaxString; (* Destination file path and name *)
    bExecute            : BOOL; (* Rising edge start fb execution *)
    tTimeOut            : TIME := DEFAULT_ADS_TIMEOUT;(* Max. ADS timeout time *)
END_VAR
VAR_OUTPUT
    bBusy           :BOOL;
(* TRUE => File copy execution in progress, FALSE => File copy execution idle *)
 bError             :BOOL;(* TRUE => Error, FALSE => No error *)
 nErrId             :UDINT;(* Error code *)
END_VAR
VAR
    fbFileOpen          :FB_FileOpen;
    fbFileClose         :FB_FileClose;
    fbFileRead          :FB_FileRead;
    fbFileWrite         :FB_FileWrite;

    hSrcFile            :UINT   := 0;(* File handle of the source file *)
 hDestFile              :UINT   := 0;(* File handle of the destination file *)

    Step                :DWORD;
    RisingEdge          :R_TRIG;

    buffRead            :ARRAY[1..1000] OF BYTE;(* Buffer *)
 cbReadLength           :UDINT := 0;
END_VAR
```

Implementation:

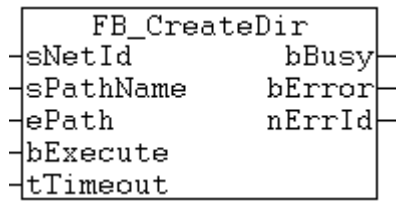
```
RisingEdge(CLK:=bExecute);

CASE Step OF
    0:      (* Idle state *)
     IF RisingEdge.Q THEN
            bBusy := TRUE;
            bError:= FALSE;
            nErrId:=0;
            Step := 1;
            cbReadLength:=0;
            hSrcFile:=0;
            hDestFile:=0;
        END_IF

    1:      (* Open source file *)
     fbFileOpen( bExecute := FALSE );
```

```
        fbFileOpen( sNetId := sSrcNetId, sPathName := sSrcPathName,
            nMode := FOPEN_MODEREAD OR FOPEN_MODEBINARY,
            ePath := PATH_GENERIC, tTimeout := tTimeOut, bExecute := TRUE );
        Step := Step + 1;
    2:
        fbFileOpen( bExecute := FALSE );
        IF NOT fbFileOpen.bBusy THEN
            IF fbFileOpen.bError THEN
                nErrId := fbFileOpen.nErrId;
                bError := TRUE;
                Step := 50;
            ELSE
                hSrcFile := fbFileOpen.hFile;
                Step := Step + 1;
            END_IF
        END_IF

    3:      (* Open destination file *)
     fbFileOpen( bExecute := FALSE );
        fbFileOpen( sNetId := sDestNetId, sPathName := sDestPathName,
            nMode := FOPEN_MODEWRITE OR FOPEN_MODEBINARY,
            ePath := PATH_GENERIC, tTimeout := tTimeOut, bExecute := TRUE );
        Step := Step+1;
    4:
        fbFileOpen( bExecute := FALSE );
        IF NOT fbFileOpen.bBusy THEN
            IF fbFileOpen.bError THEN
                nErrId := fbFileOpen.nErrId;
                bError := TRUE;
                Step := 50;
            ELSE
                hDestFile := fbFileOpen.hFile;
                Step := Step + 1;
            END_IF
        END_IF

    5:      (* Read data from source file *)
     cbReadLength := 0;
        fbFileRead( bExecute:= FALSE );
        fbFileRead( sNetId:=sSrcNetId, hFile:=hSrcFile,
            pReadBuff:= ADR(buffRead), cbReadLen:= SIZEOF(buffRead),
            bExecute:=TRUE, tTimeout:=tTimeOut );
        Step := Step + 1;
    6:
        fbFileRead( bExecute:= FALSE );
        IF NOT fbFileRead.bBusy THEN
            IF fbFileRead.bError THEN
                nErrId := fbFileRead.nErrId;
                bError := TRUE;
                Step := 50;
            ELSE
                cbReadLength := fbFileRead.cbRead;
                Step := Step + 1;
            END_IF
        END_IF

    7:      (* Write data to destination file *)
     fbFileWrite( bExecute := FALSE );
        fbFileWrite( sNetId:=sDestNetId, hFile:=hDestFile,
            pWriteBuff:= ADR(buffRead), cbWriteLen:= cbReadLength,
            bExecute:=TRUE, tTimeout:=tTimeOut );
        Step := Step + 1;
    8:
        fbFileWrite( bExecute := FALSE );
        IF NOT fbFileWrite.bBusy THEN
            IF fbFileWrite.bError THEN
                nErrId := fbFileWrite.nErrId;
                bError := TRUE;
                Step := 50;
            ELSE
                IF fbFileRead.bEOF THEN (* Check if the EOF flag ist set *)
                 Step := 50;(* Cleanup: close the destination and source files *)
             ELSE
                    Step := 5; (* Repeat reading/writing *)
             END_IF
            END_IF
        END_IF

    30:     (* Close the destination file *)
     fbFileClose( bExecute := FALSE );
```

```
        fbFileClose( sNetId:=sDestNetId, hFile:=hDestFile, bExecute:=TRUE, tTimeout:=tTimeOut );
        Step := Step + 1;
    31:
        fbFileClose( bExecute := FALSE );
        IF NOT fbFileClose.bBusy THEN
            IF fbFileClose.bError THEN
                nErrId := fbFileClose.nErrId;
                bError := TRUE;
            END_IF
            Step := 50;
            hDestFile := 0;
        END_IF

    40: (* Close source file *)
     fbFileClose( bExecute := FALSE );
        fbFileClose( sNetId:=sSrcNetId, hFile:=hSrcFile, bExecute:=TRUE, tTimeout:=tTimeOut );
        Step := Step + 1;
    41:
        fbFileClose( bExecute := FALSE );
        IF NOT fbFileClose.bBusy THEN
            IF fbFileClose.bError THEN
                nErrId := fbFileClose.nErrId;
                bError := TRUE;
            END_IF
            Step := 50;
            hSrcFile := 0;
        END_IF

    50: (* Error or ready => Cleanup *)
     IF ( hDestFile <> 0 ) THEN
            Step := 30; (* Close the destination file*)
     ELSIF (hSrcFile <> 0 ) THEN
            Step := 40; (* Close the source file *)
     ELSE
            Step := 0;      (* Ready *)
            bBusy := FALSE;
        END_IF

END_CASE
```

The complete source code for the example project can be unpacked from here: https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828003595/.exe.

3.6.13 FB_CreateDir



This functionblock creates new directories on the data medium.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId;
    sPathName   : T_MaxString;
    ePath       : E_OpenPath := PATH_GENERIC; (* Default: generic file path*)
 bExecute       : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

sPathName : The name of the new directory as string [▶ 109]. The function block can create only one new directory per call, so only the last component of sPathname can name a new directory.

ePath : The variable of this type selects one of the TwinCAT system paths [▶ 110] on the target device to create a directory.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x70C	Folder is allready existing or invalid sPathName or ePath parameter.

Example in ST:

By a rising edge at bCreate a new directory in the main directory 'C:\'named : 'PRJDATA' is created. By a rising edge at bRemove a directory with the same name can be deleted.

At bBootFolder = TRUE a driectory in the ..\TwinCAT\Boot directory can be created or deleted.

```
PROGRAM MAIN
VAR
    sFolderName : STRING := 'PRJDATA'; (* folder name *)
    bBootFolder : BOOL;

    ePath : E_OpenPath; (* folders root path *)
    sPathName : STRING;

    fbCreateDir : FB_CreateDir;
    bCreate : BOOL;
    bCreate_Busy : BOOL;
    bCreate_Error : BOOL;
    nCreate_ErrID : UDINT;

    fbRemoveDir : FB_RemoveDir;
    bRemove : BOOL;
    bRemove_Busy : BOOL;
    bRemove_Error : BOOL;
    nRemove_ErrID : UDINT;
END_VAR
```

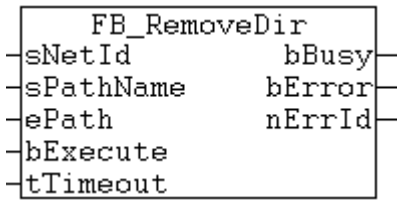
```
ePath := SEL( bBootFolder, PATH_GENERIC, PATH_BOOTPATH );
sPathName := SEL( bBootFolder, CONCAT('C:\', sFolderName), sFolderName );

IF bCreate THEN
    bCreate := FALSE;
    fbCreateDir( bExecute := FALSE );
    fbCreateDir(sNetId:= '',
        sPathName:= sPathName,
        ePath:= ePath,
        bExecute:= TRUE,
        tTimeout:= DEFAULT_ADS_TIMEOUT,
        bBusy=>bCreate_Busy, bError=>bCreate_Error, nErrId=>nCreate_ErrID );
ELSE
    fbCreateDir( bExecute := FALSE, bBusy=>bCreate_Busy, bError=>bCreate_Error, nErrId=>nCreate_ErrI
D );
```

```
END_IF



IF bRemove THEN
    bRemove := FALSE;
    fbRemoveDir( bExecute := FALSE );
    fbRemoveDir(sNetId:= '',
        sPathName:= sPathName,
        ePath:= ePath,
        bExecute:= TRUE,
        tTimeout:= DEFAULT_ADS_TIMEOUT,
        bBusy=>bRemove_Busy, bError=>bRemove_Error, nErrId=>nRemove_ErrID );
ELSE
    fbRemoveDir( bExecute := FALSE, bBusy=>bRemove_Busy, bError=>bRemove_Error, nErrId=>nRemove_ErrI
D );
END_IF
```

Requirements

Development Environment	Target System	PLC Libraries to include
TwinCAT v2.10.0 Build > 1310 (CE image v2.17d or higher)	PC or CX (x86, ARM)	TcSystem.Lib

3.6.14 FB_RemoveDir



This functionblock removes a directory from the data medium. A directory containing data can not be removed!

VAR_INPUT

```
VAR_INPUT
    sNetId          : T_AmsNetId;
    sPathName       : T_MaxString;
    ePath           : E_OpenPath := PATH_GENERIC;  (* Default: generic file path*)
    bExecute        : BOOL;
    tTimeout        : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId : Is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

sPathName : The directory [▶ 109] to be removed. The function block can remove only one directory per call, so only the last component of sPathName can name the directory to be removed.

ePath : The variable of this type selects one of the TwinCAT system paths [▶ 110] on the target device to delete a directory.

bExecute : The ADS command is triggered by a rising edge at this input.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the block has executed a command, but at the longest for the duration supplied to the 'tTimeout' input. While bBusy = TRUE, no new command will be accepted at the inputs.

bError: This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'.

nErrId: Contains the command-specific ADS error code of the most recently executed command.

Function specific ADS error code	Possible reason
0x70C	Folder not found or invalid sPathName or ePath parameter.

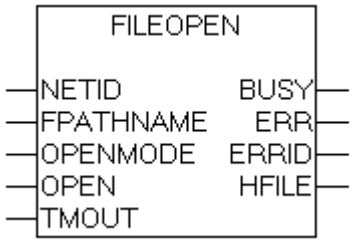
Example in ST: See description of FB_CreateDir [▶ 65].

Requirements

Development Environment	Target System	PLC Libraries to include
TwinCAT v2.10.0 Build > 1310 (CE image v2.17d or higher)	PC or CX (x86, ARM)	TcSystem.Lib

3.6.15 TwinCAT 2.7 file function blocks

3.6.15.1 FILEOPEN



With this function block a new file can be created, or a closed existing file can be re-opened for further processing.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;  (* ams net id *)
    FPATHNAME   : T_MaxString;    (* default max filename length = 255 *)
    OPENMODE    : DWORD;        (* open mode flags *)
    OPEN        : BOOL;     (* open control input *)
    TMOUT       : TIME;
END_VAR
```

NETID : is a string containing the AMS network ID [▶ 107] of the target device to which the ADS command is directed.

FPATHNAME : contains the path and file name [▶ 109] of the file to be opened.

The path can only point to the local file system of the computer. This means that network paths cannot be used here!

OPENMODE : contains the mode in which the file is to be opened. The codes listed below are the various opening modes which are already pre-defined as constants in the library and which can accordingly be passed symbolically to the function block.

- FILE_OPENCREATE: create and open a file (already existing files are overwritten with this mode!). FILE_OPENREAD: open a file for read access;
- FILE_OPENWRITE: when a file is opened for writing, the process starts at the beginning of the file.

OPEN : the ADS command is triggered by a rising edge at this input.

TMOUT : specifies the time until the abortion of the function.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    HFILE       : UINT;     (* file handle *)
END_VAR
```

BUSY : this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs.

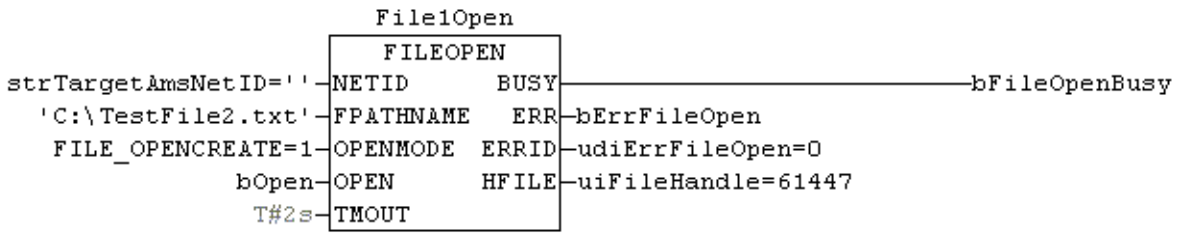
It is not the execution of the service but its acceptance whose time is monitored.

ERR : this output is set to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'ErrorId'. If the function block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : contains the instruction-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

HFILE : contains the file handle created for the file when opening has been successful.

Sample of calling the function block in FBD:



This should create (or overwrite) the file "TestFile2.txt" in the root directory of drive "C:". In the sample a proper handle is created for the file. This handle is now passed to the file function blocks described below as an identifier for the file to be processed.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.2 FILECLOSE

With this function block the file can be closed, and thereby placed into a defined state for further processing by other programs.

VAR_INPUT

```
VAR_INPUT
    NETID   : T_AmsNetId;   (* ams net id *)
    HFILE   : UINT;         (* file handle obtained through 'FILEOPEN' *)
    CLOSE   : BOOL;         (* close control input *)
    TMOUT   : TIME;
END_VAR
```

NETID : is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

HFILE : is occupied by the file handle already created by FILEOPEN.

CLOSE : the ADS command is triggered by a rising edge at this input.

TMOUT : specifies the time until the abortion of the function.

VAR_OUTPUT

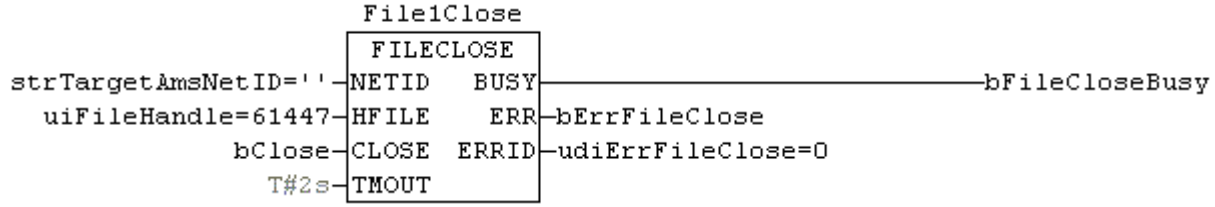
```
VAR_OUTPUT
    BUSY    : BOOL;
    ERR     : BOOL;
    ERRID   : UDINT;
END_VAR
```

BUSY : this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the function block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : contains the instruction-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Sample of calling the function block in FBD:



Here the file associated with the file handle (which was itself generated by ”File1Open") is closed again.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.3 FILEWRITE

Data can be written into a file with this function block. For write access the file must previously have been opened in the ”FILE_OPENCREATE" or ”FILE_OPENWRITE" mode, and closed again for further processing by external programs.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;   (* ams net id *)
    HFILE       : UINT;         (* file handle *)
    BUFADDR     : DWORD;        (* buffer address for write *)
    COUNT       : UDINT;        (* count of bytes for write *)
    WRITE       : BOOL;         (* write control input *)
    TMOUT       : TIME;
END_VAR
```

NETID : is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

HFILE : is occupied by the file handle already created by FILEOPEN.

BUFADDR : contains the address of the buffer containing the data to be written. The programmer is responsible for dimensioning the buffer such that it can accommodate WRITELEN bytes.

COUNT : contains the number of bytes to be written.

WRITE : the ADS command is triggered by a rising edge at this input.

TMOUT : specifies the time until the abortion of the function.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    COUNT_W     : UDINT;            (* count of bytes actually written *)
END_VAR
```

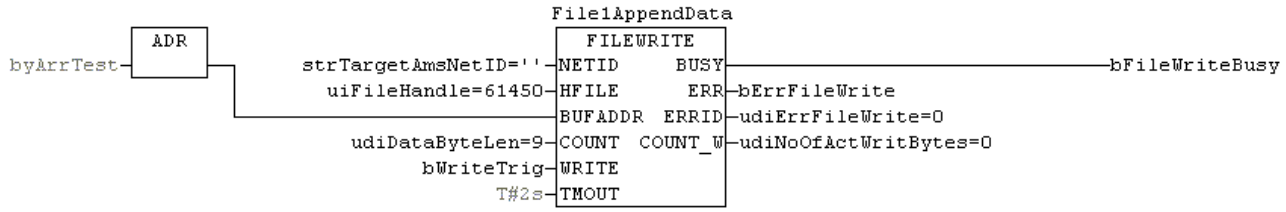
BUSY : this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the function block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : contains the instruction-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. If the error code is -1 (16#FFFF), the write operation cannot be performed because, for example, the file was not opened correctly.

COUNT_W : contains the number of bytes currently written.

Sample of calling the function block in FBD:

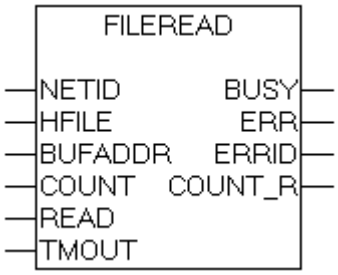


In the sample, after a rising edge at "bWriteTrig", 9 bytes of the array "byArrTest" are written at the end of the file with handle "61450".

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.4 FILEREAD



With this function block the contents of an already opened file can be read. The file must have been opened for read access in the "FILE_OPENREAD" mode.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;   (* ams net id *)
    HFILE       : UINT;         (* file handle *)
    BUFADDR     : DWORD;        (* buffer address for read *)
    COUNT       : UDINT;        (* count of bytes for read *)
    READ        : BOOL;         (* read control input *)
    TMOUT       : TIME;
END_VAR
```

NETID : is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

HFILE : is occupied by the file handle already created by FILEOPEN.

BUFADDR : contains the address of the buffer into which the data are to be read. The programmer is responsible for dimensioning the buffer such that it can accommodate WRITELEN bytes.

COUNT : contains the number of bytes to be read.

READ : the ADS command is triggered by a rising edge at this input.

TMOUT : specifies the time until the abortion of the function.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    COUNT_R     : UDINT;            (* count of bytes actually read *)
END_VAR
```

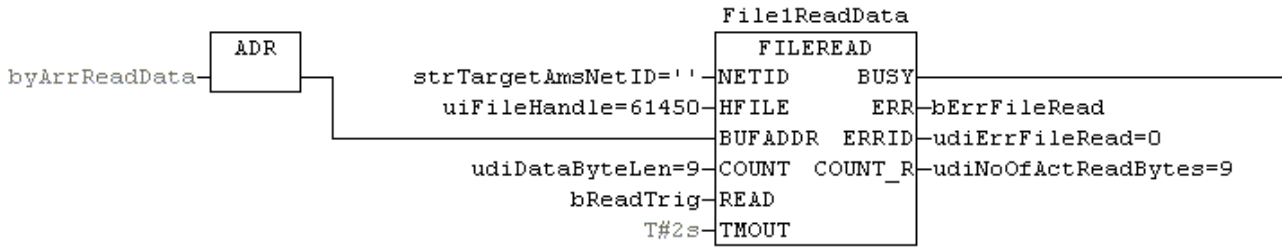
BUSY : this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the function block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : contains the instruction-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. If the error code is -1 (16#FFFF), the write operation cannot be performed because, for example, the file was not opened correctly.

COUNT_R : contains the number of currently read bytes.

Sample of calling the function block in FBD:

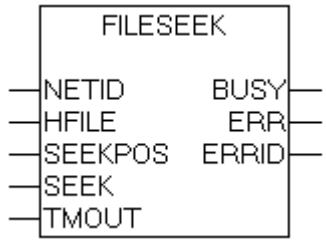


After a rising edge at ”bReadTrig" and successful execution of the read instruction the currently read bytes from the file are found in ”byArrReadData". The number of bytes actually read in the previous read procedure can be determined from the parameter ”COUNT_R". The sample shows that nine bytes were read.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.5 FILESEEK



With this function block the file pointer of an opened file can be set to a definable position.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;   (* ams net id *)
    HFILE       : UINT;      (* file handle *)
    SEEKPOS     : UDINT;         (* new seek pointer position *)
    SEEK        : BOOL;      (* seek control input *)
    TMOUT       : TIME;
END_VAR
```

NETID : is a string containing the AMS network identifier [▶ 107] of the target device to which the ADS command is directed.

HFILE : is occupied by the file handle already created by FILEOPEN.

SEEKPOS : contains the desired (absolute) target position of the file pointer. The constants FILE_SEEKEND and FILE_SEEKBEGIN can be used to set the file pointer to the end or to the beginning of the file.

SEEK : the ADS command is triggered by a rising edge at this input.

TMOUT : specifies the time until the abortion of the function.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

ERR : this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the function block has a timeout error, 'Error' is TRUE and 'ErrorId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

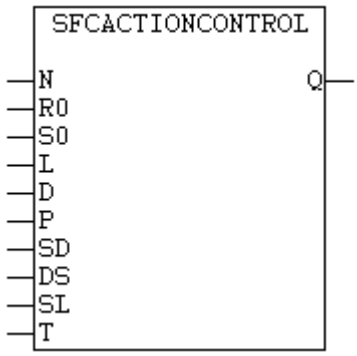
ERRID : contains the instruction-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7 IEC steps / SFC flags function blocks

3.7.1 SFCActionControl

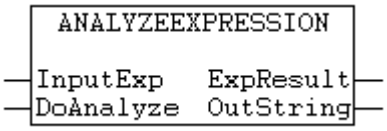


This function is required to use IEC steps in SFC programs / projects. Only the library with the FB must be included to the projec, but no instances are reqiured.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	IecSfc.Lib
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7.2 AnalyzeExpression

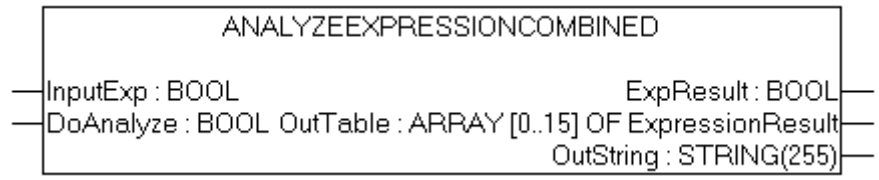


This function block is required in PLC projects, which use the SFC flags. No instances were created. Only the corresponding PLC library must be included to the project.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Analyzation.Lib
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7.3 AnalyzeExpressionCombined

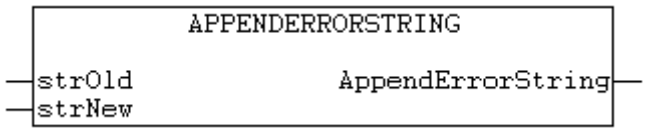


The function block is needed in PLC projects that uses sfc flags.No instances are created. The corresponding PC library has to be included in the project.

Requirements

Development Environment	Target system type	PC Libraries to be linked
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7.4 AppendErrorString



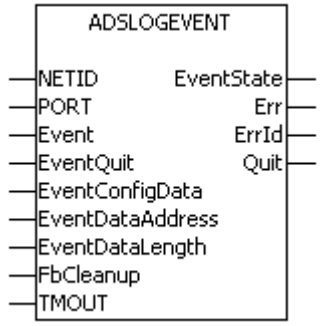
This function is required in PLC projects, which use the SFC flags. The function must not be called in the project. Only the corresponding PLC library must be included to the project.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Analyzation.Lib
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.8 Eventlogger function blocks

3.8.1 ADSLOGEVENT



The function block permits the sending and the acknowledgement of messages to the TwinCAT Eventlogger.

VAR_INPUT

```
VAR_INPUT
    NETID           : STRING(23);
    PORT             : UINT;
    Event            : BOOL;
    EventQuit        : BOOL;
    EventConfigData  : TcEvent;
    EventDataAddress : UDINT;
    EventDataLength  : UDINT;
    FbCleanup        : BOOL;
    TMOUT            : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

PORT : Contains the port number of the ADS device. The TwinCAT Eventlogger has the port number 110.

Event : With a rising edge the "coming" of the event is signaled, with a falling edge the "going" of the event.

EventQuit : With a rising edge the event is acknowledged.

EventConfigData : Data structure in the event parameters [▶ 112].

EventDataAddress : Address with the data, which sould be sent with the event.

EventDataLength : Length of the data, which should be sent with the event.

FbCleanup : At TRUE the function block is initialised completely.

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

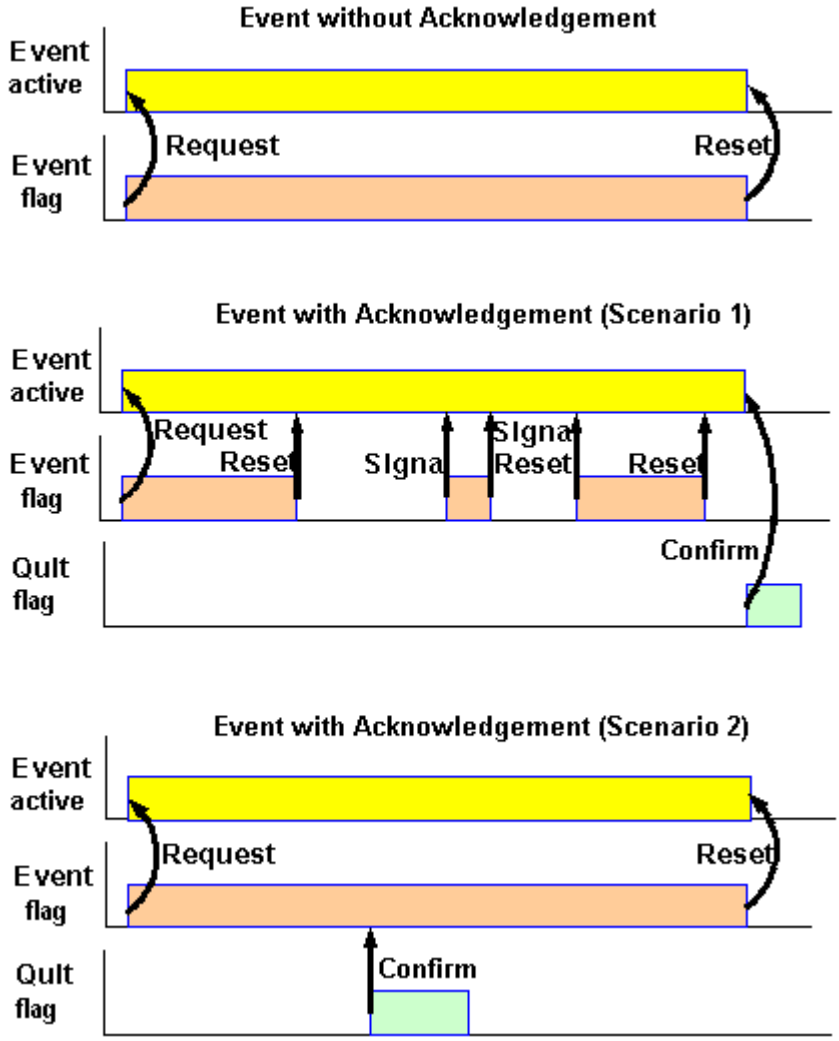
```
VAR_OUTPUT
    EventState : UDINT;
    Err        : BOOL;
    ErrId      : UDINT;
    Quit       : BOOL;
END_VAR
```

EventState : State of the events.

ERR : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘ErrorId’. If the block has a timeout error, ‘Error’ is TRUE and ‘ErrorId’ is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

ERRID : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

Quit : The Event is acknowledged.



The picture above shows the fundamental construction.

At non obligatory acknowledge messages, the event is reported with the rising edge at the event input of the function block and therewith active in the Eventlogger. The falling edge at the event input triggers the reset. With this signal the event is cancelled again in the Eventlogger.

At obligatory acknowledge messages the event is again activated wtih the rising edge at the event input. The event can be deactivated

* with a falling edge at the event input (if before an acknowledgement signal has come from the PLC with the quit input, or from the visualisation) or

* with a rising edge at the quit input (if before a reset was released by a falling edge at the event input)

If a reset occurs between event activation and occurence of the acknowledgement, the next occurence of the event input is called "signal". Therewith a request is reported at already active event.

Stepwise flow:

- **Configurate an Event:**

  Parameterise EventConfigData [▶ 112] sructure.

- **transfer of parameter**
  Create an address to a structure, an array or a single variable with ADS to EventDataAddress.
  Determine the length of the structure, the array or the single variable with the SIZEOF operator, and

supply the length to the input EventDataLength. If for example a structrue with an INT and LREAL variable should be transferred with an event, a structure with these two components has to be created and instantiated. The address and the length of this instance must be transferred.

- **Set an event:**
  Rising edge at the event input
- **Reset an event:**
  Falling edge at the event input
- **Acknowledge an event:**
  Rising edge at the quit input
- **Complete delete of the instance:**
  With the rising edge at the input FbCleanup the content of the instance is completely deleted. Therewith not straight an existing event is deleted from the Eventlogger.

After an event was sent to the Eventlogger, the Status of the event [▶ 115] changes visibly at the eventstate output.

Example:

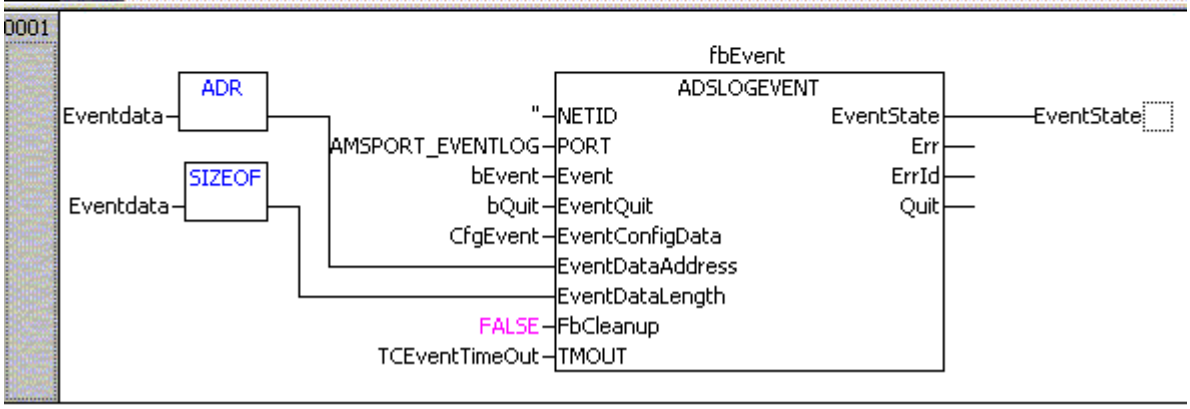
Configuration of the event:

```
PROGRAM Config
VAR
    CfgEvent : TcEvent;
    TCEventDataFormatString        :    STRING := '%f%d';
END_VAR
```

```
CfgEvent.Class :=TCEVENTCLASS_ALARM;
CfgEvent.Prio := 2;
CfgEvent.Id := i;
CfgEvent.bQuitRequired := FALSE;
CfgEvent.DataFormatStrAddress := ADR(TCEventDataFormatString);
CfgEvent.UserFlags := 16#0000_0000;
CfgEvent.Flags := TCEVENTFLAG_LOG OR TCEVENTFLAG_MSGBOX OR TCEVENTFLAG_SRCID;
CfgEvent.StreamType := TCEVENTSTREAM_SIMPLE;
CfgEvent.SourceString := '';
CfgEvent.SourceId := 100;
CfgEvent.ProgId := 'TcEventLogger.TcLogFormatter';
```

Call FB ADSLOGEVENT:

```
0001 PROGRAM MAIN_DOKU
0002 VAR
0003     fbEvent: ADSLOGEVENT;
0004     CfgEvent : TcEvent;
0005     Eventdata : ParaStruct;
0006     EventState : UDINT;
0007     bEvent : BOOL;
0008     bQuit : BOOL;
0009 END_VAR
0010 VAR CONSTANT
0011     TCEventDataFormatString     :   STRING := '%f%d';
0012     TCEventTimeOut              :   TIME := t#1s;
0013 END_VAR
0014
```



https://infosys.beckhoff.com/content/1033/tcplclibsystem/Resources/11828005003/.zip

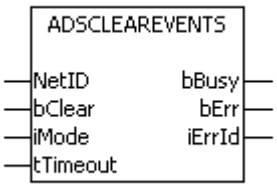
The sample consists of

- An Event Configuration.ecp which can be opened and activated using the TcEventConfigurator
- A PLC sample Program

Requirements

Development environment	Target system type	PLC libraires to include
TwinCAT v2.7.0	PC or CX (x86)	PLCEvent.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.8.2 ADSCLEAREVENTS



The function block permits the sending and the acknowledge of messages to the TwinCAT Eventlogger.

VAR_INPUT

```
VAR_INPUT
    NETID    : STRING(23);
    bClear   : BOOL;
    iMode    : UDINT;
    tTimeout : TIME;
END_VAR
```

NETID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

iMode : Mode to delete the events. Defined in the enum E_TcEventClearModes [▶ 111].

TMOUT : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy :  BOOL;
    bErr :   BOOL;
    iErrId : UDINT;
END_VAR
```

bBusy : Is TRUE as long as the action is executed. In this time no new command is possible.

bErr : This output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in ‘iErrorId’. If the block has a timeout error, ‘bErr’ is TRUE and ‘iErrorId’ is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

iErrId : Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

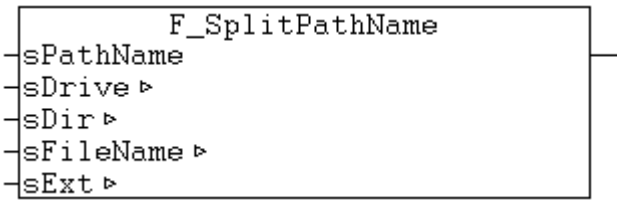
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4 Functions

4.1 General Functions

4.1.1 F_SplitPathName



This function splits a complete path name into its four components. These are stored in the strings named *sDrive*, *sDir*, *sFileName* and *sExt.*

FUNCTION F_SplitPathName : BOOL

```
VAR_INPUT
    sPathName : T_MaxString;
END_VAR
```

sPathName: Complete file name as string [▶ 109]: 'X:\DIR\SUBDIR\FILENAME.EXT'.

```
VAR_IN_OUT
    sDrive    : STRING(3);
    sDir      : T_MaxString;
    sFileName : T_MaxString;
    sExt      : T_MaxString;
END_VAR
```

sDrive: Drive name with a double point ('C:', 'A:' etc.).

sDir: Directory name inlusive the leading and ending backslash ('\BC \INCLUDE\', '\SOURCE\' etc.).

sFileName: File name.

sExt: Contains the point that starts the name extension ('.C', '.EXE' etc.).

Return parameter	Description
TRUE	No error
FALSE	Error. Check the function parameter.

Example for a call in ST:

The path name: 'C:\TwinCAT\Plc\Project01\Data.txt' is splitted in the following components:

sDrive: = 'C:'

sDir: '\TwinCAT\Plc\Project01\'

sFileName: 'Data'

sExt: '.txt'

```
PROGRAM MAIN
VAR
    bSplit      : BOOL;
    sPathName   : T_MaxString := 'C:\TwinCAT\Plc\Project01\Data.txt';
    sDrive      : STRING(3);
    sDir        : T_MaxString;
    sFileName   : T_MaxString;
```

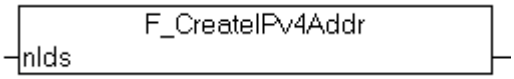
```
    sExt        : T_MaxString;
    bSuccess    : BOOL;
END_VAR
```

```
IF bSplit THEN
    bSplit := FALSE;
    bSuccess := F_SplitPathName( sPathName := sPathName,
                sDrive := sDrive,
                sDir := sDir,
                sFileName := sFileName,
                sExt := sExt );
END_IF
```

Requirements

Development Environment	Target System	PLC Libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcSystem.Lib

4.1.2 F_CreateIPv4Addr



The function F_CreateIPv4Addr returns formatted (IPv4) Internet Protocol network address string (e.g.: '172.16.7.199').

FUNCTION F_CreateIPv4Addr : T_IPv4Addr

T_IPv4Addr [▶ 109]

```
VAR_INPUT
    nIds    : T_IPv4AddrArr;
END_VAR
```

nIds: Byte array [▶ 109]. Every byte is equivalent to one number of the (IPv4) Internet Protocol network adress. The address bytes are represented in network byte order.

Example in structured text:

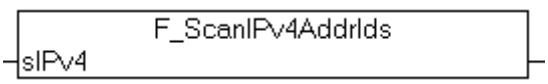
```
PROGRAM MAIN
VAR
    ids     : T_IPv4AddrArr := 172, 16, 7, 199;
    sIPv4   : T_IPv4Addr := '';
END_VAR
```

```
sIPv4 := F_CreateIPv4Addr( ids );(* Result: '172.16.7.199' *)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1340	PC or CX (x86, ARM)	TcSystem.Lib

4.1.3 F_ScanIPv4AddrIds



The function F_ScanIPv4AddrIds converts a string with the (IPv4) Internet Protocol network address into single address bytes. The single address bytes are converted from left to right. They are returned as an array of bytes. The address bytes are represented in network byte order.

FUNCTION F_ScanIPv4AddrIds: T_IPv4AddrArr

T_IPv4AddrArr [▶ 109]

```
VAR_INPUT
    sIPv4 : T_IPv4Addr;
END_VAR
```

sIPv4: Internet Protocol network address [▶ 109] as string. E.g.: '172.16.7.199'.

Input value	Return value	Description
sIPv4 ≠ '' (empty string) and sIPv4 ≠ '0.0.0.0'	All bytes are zero	Error during conversion. Please check the format of sIPv4 input string.

Example in structured text:

Internet Protocol (IPv4) network address string: '172.16.7.199' is converted to an array of address bytes.

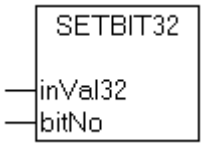
```
PROGRAM MAIN
VAR
    ids     : T_IPv4AddrArr;
    sIPv4   : T_IPv4Addr := '172.16.7.199';
END_VAR
```

```
ids := F_ScanIPv4AddrIds( sIPv4 ); (* Result: ids[0]:=172, ids[1]:=16, ids[2]:=7, ids[3]:=199 *)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1340	PC or CX (x86, ARM)	TcSystem.Lib

4.1.4 SETBIT32



The function sets the bit specified by a bit number in the 32 bit value that is passed to it and returns the resulting value.

FUNCTION SETBIT32 : DWORD

```
VAR_INPUT
    inVal32     :DWORD;
    bitNo       :SINT;
END_VAR
```

inVal32: The 32-bit value that is to be modified;

bitNo: The number of the bit that is to be set (0-31). This number is internally converted to a modulo 32 value prior to execution;

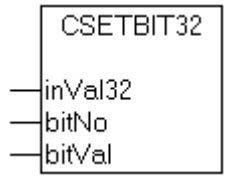
Example of calling the function in FBD:

This sets bit 31 of the input value '0'. The result is the (hex) value '80000000'.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.5 CSETBIT32



The function sets/resets the bit specified by a bit number in the 32 bit value that is passed to it and returns the resulting value.

FUNCTION CSETBIT32 : DWORD

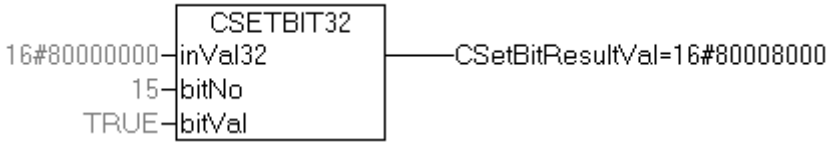
```
VAR_INPUT
    inVal32     :DWORD;
    bitNo       :SINT;
    bitVal      :BOOL;
END_VAR
```

inVal32: The 32-bit value that is to be modified;

bitNo: The number of the bit that is to be set/reset (0-31). This number is internally converted to a modulo 32 value prior to execution;

bitVal: The new value of the set/reset bit (TRUE = 1, FALSE = 0);

Example of calling the function in FBD:



This sets bit 15 of the input value '16#8000000' to 1. The result is the (hex) value '80008000'.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.6 GETBIT32

The function returns the status of the bit specified by a bit number in the 32 bit value that is passed to it as a boolean resulting value. The input value is not altered.

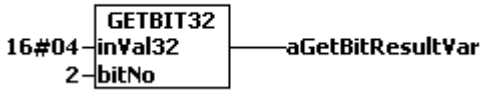
FUNCTION GETBIT32 : BOOL

```
VAR_INPUT
    inVal32     :DWORD;
    bitNo       :SINT;
END_VAR
```

inVal32: The 32-bit value that is to be modified;

bitNo: The number of the bit that is to be set (0-31). This number is internally converted to a modulo 32 value prior to execution;

Example of calling the function in FBD:

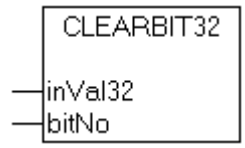


Bit 2 in the input value '04' is interrogated here, and assigned to the boolean variable 'aGetBitResultVar'. When examined, it is found in this example to be 'TRUE'.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.7 CLEARBIT32



The function resets the bit specified by a bit number in the 32 bit value that is passed to it to zero and returns the resulting value.

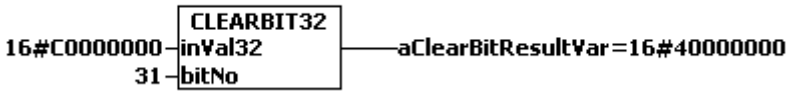
FUNCTION CLEARBIT32 : DWORD

```
VAR_INPUT
    inVal32     :DWORD;
    bitNo       :SINT;
END_VAR
```

inVal32: The 32-bit value that is to be modified;

bitNo: The number of the bit that is to be set (0-31). This number is internally converted to a modulo 32 value prior to execution;

Example of calling the function in FBD:

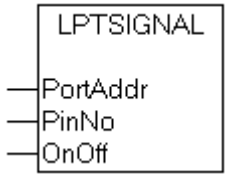


This resets bit 31 of the input value 'C0000000'. The result is the (hex) value '40000000'.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.8 LPTSIGNAL



This function sets a defined output bit in a Centronics interface to a logical high or low level, and can, for example, be used for run-time measurements with an oscilloscope. The function returns the preset state which has been written to the output pin.

FUNCTION LPTSIGNAL: BOOL

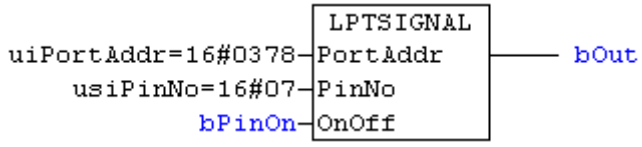
```
VAR_INPUT
    PortAddr    :UINT;
    PinNo       :INT;
    OnOff       :BOOL;
END_VAR
```

PortAddr: Address of the port which is available for the desired LPT interface;

PinNo: Contains the number of the pin (Pin 0 .. 7) which is to be written by the PLC;

OnOff: Contains the state which is to be written to that pin;

Example of calling the function in FBD:

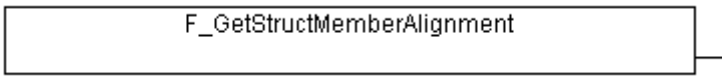


In the example, bit 7 of port 378 (hex) is set to 1.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.9 F_GetStructMemberAlignment



The function returns information about used data struct member alignment setting. The alignment is affecting the the way data structure elements are arranged in computer memory.

FUNCTION F_GetStructMemberAlignment : BYTE

```
VAR_INPUT
    (* none input parameters *)
END_VAR
```

Return value	Description
1	1 byte alignment (e.g. TwinCAT v2.11, x86 target platform)
2	2 byte alignment
4	4 byte alignment (e.g. TwinCAT v2.11, ARM target platform)
8	8 byte alignment

The following examples show the arrangement of the data structure elements in the memory, depending on the memory alignment employed.

?? := Padding byte

Example 1

```
TYPE ST_TEST1
STRUCT
    ui8 : BYTE      := 16#FF;(* FF *)
    f64 : LREAL     := 1234.5678;(* AD FA 5C 6D 45 4A 93 40 *)
END_STRUCT
END_TYPE
```

```
test1 : ST_TEST1;
```

Alignment	SIZEOF(test1)	Memory contents
1 byte	**9**	**FF AD FA 5C 6D 45 4A 93 40**
2 byte	**10**	**FF ?? AD FA 5C 6D 45 4A 93 40**
4 byte	**12**	**FF ?? ?? ?? AD FA 5C 6D 45 4A 93 40**
8 byte	**16**	**FF ?? ?? ?? ?? ?? ?? ?? AD FA 5C 6D 45 4A 93 40**

Example 2

Converting the order of the structure elements changes the arrangement of the padding bytes. These are now added at the end.

```
TYPE ST_TEST2
STRUCT
    f64 : LREAL     := 1234.5678;(* AD FA 5C 6D 45 4A 93 40 *)
    ui8 : BYTE      := 16#FF;(* FF *)
END_STRUCT
END_TYPE
```

```
test2 : ST_TEST2;
```

Alignment	SIZEOF(test2)	Memory contents
1 byte	9	AD FA 5C 6D 45 4A 93 40 FF
2 byte	10	AD FA 5C 6D 45 4A 93 40 FF ??
4 byte	12	AD FA 5C 6D 45 4A 93 40 FF ?? ?? ??
8 byte	16	AD FA 5C 6D 45 4A 93 40 FF ?? ?? ?? ?? ?? ?? ?? ??

Example 3

In the case of 2, 4 and 8-byte alignment, the elements ui32 and f64 are already suitably aligned, so that no padding bytes need to be added.

```
TYPE ST_TEST3
STRUCT
    ui8     : BYTE  := 16#FF;(* FF *)
    ui16    : WORD  := 16#1234;(* 34 12 *)
    ui32    : DWORD := 16#AABBCCDD;(* DD CC BB AA *)
    f64     : LREAL := 1234.5678;(* AD FA 5C 6D 45 4A 93 40 *)
END_STRUCT
END_TYPE
```

```
test3 : ST_TEST3;
```

Alignment	SIZEOF(test3)	Memory contents
1 byte	15	FF 34 12 DD CC BB AA AD FA 5C 6D 45 4A 93 40
2 byte	16	FF ?? 34 12 DD CC BB AA AD FA 5C 6D 45 4A 93 40
4 byte	16	FF ?? 34 12 DD CC BB AA AD FA 5C 6D 45 4A 93 40
8 byte	16	FF ?? 34 12 DD CC BB AA AD FA 5C 6D 45 4A 93 40

Example 4

```
TYPE ST_A1
STRUCT
    ui8     : BYTE  := 16#FF;(* FF *)
    ui32    : DWORD := 16#AABBCCDD;(* DD CC BB AA *)
    rsv     : BYTE  := 16#EE;(* EE *)
END_STRUCT
END_TYPE

TYPE ST_A2
STRUCT
    ui16    : WORD  := 16#1234;(* 34 12 *)
    ui8     : BYTE  := 16#55;(* 55 *)
END_STRUCT
END_TYPE

TYPE ST_TEST4
STRUCT
    a1 : ST_A1;
    a2 : ST_A2;
END_STRUCT
END_TYPE
```

```
test4 : ST_TEST4;
```

Align-ment	SIZEOF(test4)	SIZEOF(test4.a1)	a1/a2 pad-ding bytes	SIZEOF(test4.a2)	Memory contents
1 byte	9	6	-	3	FF DD CC BB AA EE 34 12 55
2 byte	12	8	-	4	FF ?? DD CC BB AA EE ?? 34 12 55 ??
4 byte	16	12	-	4	FF ?? ?? ?? DD CC BB AA EE ?? ?? ?? 34 12 55 ??
8 byte	16	12	-	4	FF ?? ?? ?? DD CC BB AA EE ?? ?? ?? 34 12 55 ??

Example 5

```
TYPE ST_D1
STRUCT
    ui16    : WORD  := 16#1234;(* 34 12 *)
    ui8     : BYTE  := 16#55;(* 55 *)
END_STRUCT
END_TYPE

TYPE ST_D2
STRUCT
    ui8     : BYTE  := 16#FF;(* FF *)
    f64     : LREAL := 1234.5678;(* AD FA 5C 6D 45 4A 93 40 *)
    rsv     : BYTE  := 16#EE;(* EE *)
END_STRUCT
END_TYPE

TYPE ST_TEST5
STRUCT
    d1 : ST_D1;
    d2 : ST_D2;
END_STRUCT
END_TYPE
```

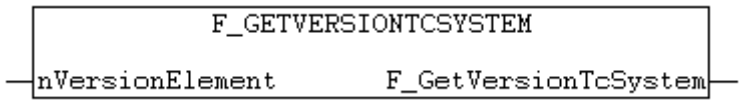
```
test5 : ST_TEST5;
```

Align-ment	SIZEOF(test 5)	SIZEOF(test5 .d1)	d1/d2 padding bytes	SIZEOF(test 5.d2)	Memory contents
1 byte	13	3	-	10	34 12 55 FF AD FA 5C 6D 45 4A 93 40 EE
2 byte	16	4	-	12	34 12 55 ?? FF ?? AD FA 5C 6D 45 4A 93 40 EE ??
4 byte	20	4	-	16	34 12 55 ?? FF ?? ?? ?? AD FA 5C 6D 45 4A 93 40 EE ?? ?? ??
8 byte	32	4	4	24	34 12 55 ?? ?? ?? ?? ?? FF ?? ?? ?? ?? ?? ?? ?? AD FA 5C 6D 45 4A 93 40 EE ?? ?? ?? ?? ?? ?? ??

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11 Build > 1553 TwinCAT v2.11 R2 Build > 2034	PC or CX (x86, ARM)	TcSystem.Lib

4.1.10 F_GetVersionTcSystem



The function returns library version info.

FUNCTION F_GetVersionTcSystem : UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

nVersionElement : Version parameter:

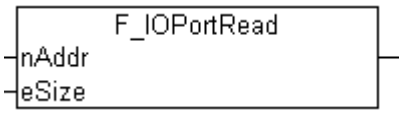
- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.2 I/O port access

4.2.1 F_IOPortRead



Most times a digital I/O port is a one byte wide I/O Position, which is mapped in the memory or as port. If you write a value at this place, the electrical signal at the output pins is changed according to the written bits.If you read a value from the input positon, the current logistic level is returned at the input pins as individual bit values.

The function F_IOPortRead reads a one eSize wide I/O position. The read value is returned from the function as return value. See description of the function F_IOPortWrite [▶ 91]

FUNCTION F_IOPortRead : DWORD

```
VAR_INPUT
    nAddr : UDINT;
    eSize : E_IOAccessSize;
END_VAR
```

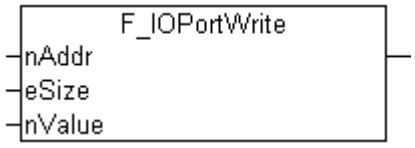
nAddr: I/O Port address.

eSize: Number [▶ 112] of data byte to be read.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcSystem.Lib

4.2.2 F_IOPortWrite



The function F_IOPortWrite writes a one eSize wide I/O position. See also description F_IOPortRead [▶ 91] Function.

Direct hardware access is no problem if you only read data.

NOTE

Risk of data loss using write access

HOWEVER, BY WRITING DATA, YOU MAY CRASH OR EVEN PERMANENTLY DAMAGE YOUR HARDWARE AND/OR DESTROY DATA ON YOUR DISKS. THIS FUNCTION MAY SEVERELY DAMAGE YOUR HARDWARE AND CAN EVEN MAKE YOUR COMPUTER UNBOOTABLE. PLEASE DO ONLY USE IT, IF YOU EXACTLY KNOW WHAT YOU ARE DOING!

FUNCTION F_IOPortWrite : BOOL

```
VAR_INPUT
    nAddr : UDINT;
    eSize : E_IOAccessSize;
    nValue: DWORD;
END_VAR
```

nAddr: I/O port address.

eSize: Number [▶ 112] of data bytes to write.

nValue: Value, that shall be written.

Return paramters	Description
TRUE	Kein Fehler
FALSE	Fehler

ST sample:

In the following example, a PLC module for direct control of the PC speaker was implemented using the I/O port functions.Interface:

```
FUNCTION_BLOCK FB_Speaker
(* Sample code from: "PC INTERN 2.0", ISBN 3-89011-331-1, Data Becker *)
VAR_INPUT
    freq : DWORD := 10000; (* Frequency [Hz] *)
 tDuration : TIME := T#1s; (* Tone duration *)
    bExecute : BOOL; (* Rising edge starts function block execution *)
END_VAR
VAR_OUTPUT
    bBusy : BOOL;
    bError : BOOL;
    nErrID : UDINT;
END_VAR
VAR
    fbTrig : R_TRIG;
    nState : BYTE;
    sts61H : DWORD;
    cnt42H : DWORD;
    cntLo : DWORD;
    cntHi : DWORD;
    timer : TON;
END_VAR
```

Implementation:

```
fbTrig( CLK := bExecute );
CASE nState OF
0:
IF fbTrig.Q THEN
    bBusy := TRUE;
    bError := FALSE;
    nErrID := 0;
    timer( IN := FALSE );

    IF F_IOPortWrite( 16#43, NoOfByte_Byte, 182 ) THEN
```

```
        cnt42H := 1193180 / freq;
        cntLo := cnt42H AND 16#FF;
        cntHi := SHR( cnt42H, 8 ) AND 16#FF;

        F_IOPortWrite( 16#42, NoOfByte_Byte, cntLo ); (* LoByte *)
        F_IOPortWrite( 16#42, NoOfByte_Byte, cntHi ); (* HiByte *)

        timer( IN := TRUE, PT := tDuration );

        sts61H := F_IOPortRead( 16#61, NoOfByte_Byte );
        sts61H := sts61H OR 2#11;
        F_IOPortWrite( 16#61, NoOfByte_Byte, sts61H ); (* speaker ON *)

        nState := 1;
    ELSE
        nState := 100;
    END_IF
END_IF

1:
timer( );
IF timer.Q THEN

    sts61H := F_IOPortRead( 16#61, NoOfByte_Byte );
    sts61H := sts61H AND 2#11111100;
    F_IOPortWrite( 16#61, NoOfByte_Byte, sts61H ); (* speaker off *)
    bBusy := FALSE;
    nState := 0;
END_IF

100:
bBusy := FALSE;
bError := TRUE;
nErrID := 16#8000;
nState := 0;

END_CASE
```

Test application:

```
PROGRAM MAIN
VAR
    fbSpeaker : FB_Speaker;
    bStart : BOOL;
END_VAR
```

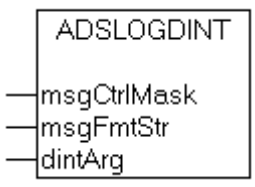
```
fbSpeaker( freq:= 5000,
tDuration:= t#1s,
bExecute:= bStart );
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcSystem.Lib

4.3 ADS Functions

4.3.1 ADSLOGDINT



This function issues when called a message box holding a specifiable text on the screen, and writes an entry into the system's log.
Since a PLC program is cyclically processed, it is necessary for an item such as a message box to be output under edge-control. This is most easily achieved with an R_TRIG or F_TRIG function block placed in series (see also examples below).
Using the ADSLOGDINT function a DINT value (4 byte signed integer) can be inserted in the text to be output at a point specified by the user. For this purpose, the stored format string must contain the characters '%d' at the desired location. The result value contains the function error code, or, if successful, 0.

FUNCTION ADSLOGDINT : DINT

```
VAR_INPUT
    msgCtrlMask     : DWORD;
    msgFmtStr       : T_MaxString;
    dintArg         : DINT;
END_VAR
```

msgCtrlMask : Control mask which determines the type and effect of the message output (see separate table).

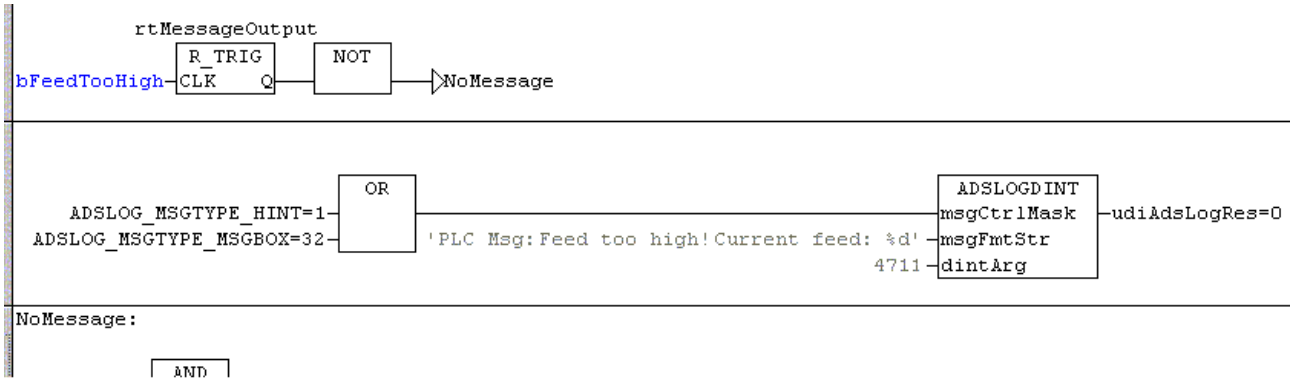
Constant	Description
ADSLOG_MSGTYPE_HINT	Message type is advice.
ADSLOG_MSGTYPE_WARN	Message type is warning.
ADSLOG_MSGTYPE_ERROR	Message type is error.
ADSLOG_MSGTYPE_LOG	Message is written into the log.
ADSLOG_MSGTYPE_MSGBOX	Message is output to a message box.
ADSLOG_MSGTYPE_RESOURCE	Message is fetched from a resource file. (not currently supported)
ADSLOG_MSGTYPE_STRING	Message is a directly given string (default).

The control masks can be ORed in the desired combination.

msgFmtStr : Contains the message [▶ 109] to be issued. It can contain the formatting code '%d' for the output of a DINT value at any position.

dintArg : Contains the numerical value to be inserted into the message.

Example of calling the function in FBD:



The resulting message box

The DINT value 4711 is inserted here into a message. The insertion point is marked by the %d characters in the format string.

Example of calling the function in ST:

```
rtMessageOutput: R_TRIG; (* Declaration*)
bFeedTooHigh: BOOL;
udiAdsLogRes: UDINT;
(*--------------------------------------*)
rtMessageOutput(CLK := bFeedTooHigh);
IF rtMessageOutput.Q THEN
   UdiAdsLogRes := ADSLOGDINT( msgCtrlMask :=
ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_MSGBOX,
       msgFmtStr := 'PLC
Msg: Feed too high! Current feed: %d',
        dintArg:= 4711);
END_IF;
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.2 ADSLOGLREAL

When called, the function outputs a message box with a predefinable text to the screen and writes an entry to the system's event log. In the text to be output, a LREAL value (floating-point number) can be inserted at a point specified by the user. For this purpose, the created format string must contain the string '%f' at the desired position. Please keep in mind that also here, as shown in the sample, the function must be called using edge-control (see also description ADSLOGDINT [▶ 93]). The return parameter contains the function error code, or 0 if successful.

FUNCTION ADSLOGLREAL : DINT

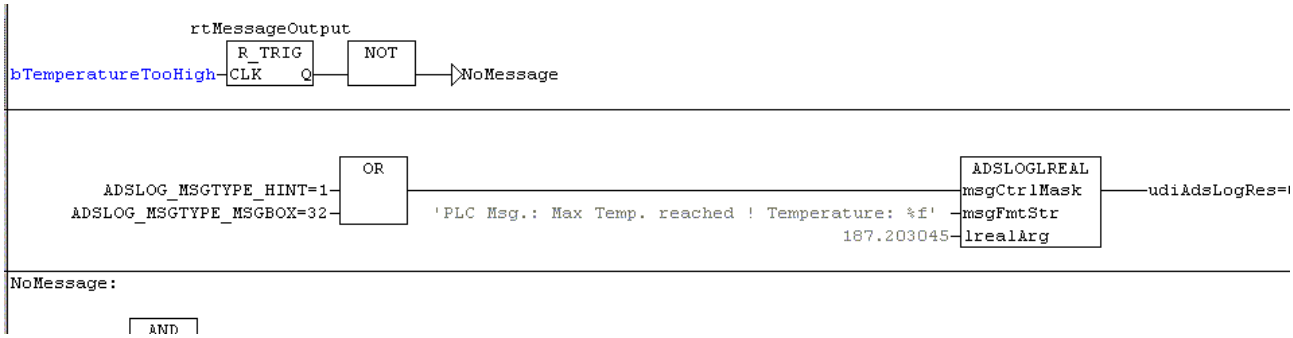
```
VAR_INPUT
    msgCtrlMask     : DWORD;
    msgFmtStr       : T_MaxString;
    lrealArg        : LREAL;
END_VAR
```

msgCtrlMask : Control mask which determines the type and effect of the message output. Listing of all the control masks for message output currently implemented as global constants in the library (see description of the function ADSLOGDINT [▶ 93]).

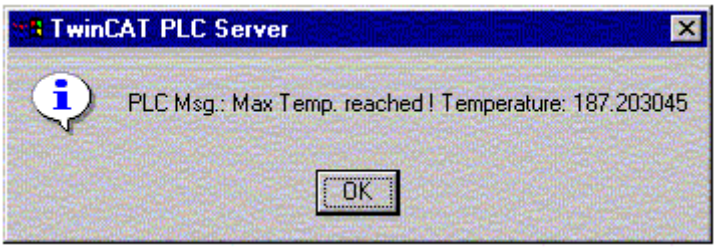
msgFmtStr : Contains the message [▶ 109] to be issued. It can contain the formatting code ‘%d’ for the output of a DINT value at any position.

lrealArg : Contains the numerical value to be inserted into the message.

Example of calling the function in FBD:



The resulting message box



Here the LREAL value 187.203045 is inserted into a message. The insertion point is marked by the '%f' characters in the format string. The number is truncated after the sixth decimal point during output.

Example of calling the function in ST:

```
rtMessageOutput: R_TRIG; (* Declaration*)
bTemperatureTooHigh: BOOL;
udiAdsLogRes: UDINT;
(*--------------------------------------*)
rtMessageOutput(CLK := bTemperatureTooHigh);
IF rtMessageOutput.Q THEN
    udiAdsLogRes := ADSLOGLREAL( msgCtrlMask :=
ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_MSGBOX,
       msgFmtStr := 'PLC
Msg.: Max Temp. reached ! Temperature: %f', lrealArg :=
187.203045);
END_IF;
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.3 ADSLOGSTR

When called, the function outputs a message box with a predefinable text to the screen and writes an entry to the system's event log. A string can be inserted into the text to be output at a position specified by the user. For this purpose, the created format string must contain the string '%s' at the desired position.

Here, as also shown in the sample, the function must be called using edge-control (see also description ADSLOGDINT [▶ 93]).

The return parameter contains the function error code, or 0 if successful.

FUNCTION ADSLOGSTR : DINT

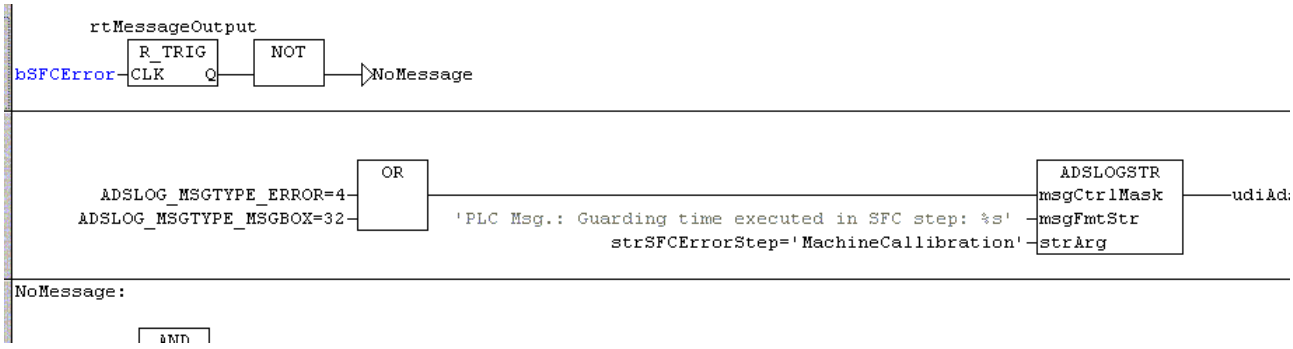
```
VAR_INPUT
    msgCtrlMask : DWORD;
    msgFmtStr   : T_MaxString;
    strArg      : T_MaxString;
END_VAR
```

msgCtrlMask : Control mask which determines the type and effect of the message output (see separate table: ADSLOGDINT [▶ 93]).

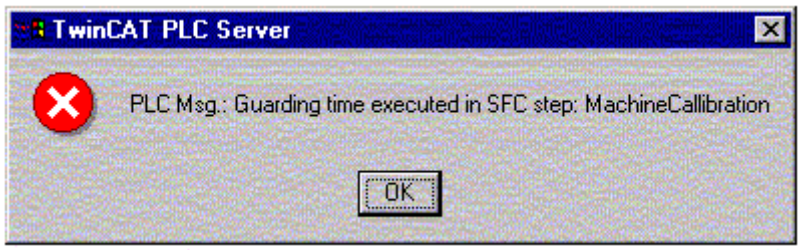
msgFmtStr : Contains the message [▶ 109] to be issued. It can contain the formatting code '%d' for the output of a DINT value at any position.alten.

strArg : Contains the string which is to be inserted into the message.

Example of calling the function in FBD:



The resulting message box



With this, the PLC programmer inserts the string stored in the variable 'strSFCErrorStep' into the message. The insertion point is marked by the %s characters in the format string.

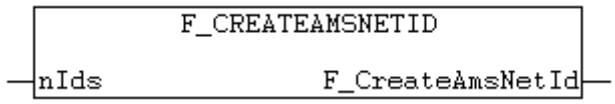
Example of calling the function in ST:

```
strSFCErrorStep : STRING; (* Declaration*)
rtMessageOutput: R_TRIG;
bSFCError: BOOL;
(*-------------------------------------*)
rtMessageOutput(CLK := bSFCError);
IF rtMessageOutput.Q THEN
    udiAdsLogRes := ADSLOGSTR( msgCtrlMask :=
ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_MSGBOX,
        msgFmtStr := 'PLC Msg.:
Guarding time executed in SFC step: %s', strArg :=
strSFCErrorStep);
END_IF;
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.4 F_CreateAmsNetId



The F_CreateAmsNetId function returns formatted AmsNetId string (e.g.: '127.16.17.3.1.1').

FUNCTION F_CreateAmsNetId : T_AmsNetId

T_AmsNetId [▶ 107]

```
VAR_INPUT
    nIds    : T_AmsNetIdArr;
END_VAR
```

nIds : Byte array [▶ 108]. Every byte is equivalent to one number of the network adress. The address bytes are represented in network byte order.

Example in structured text:

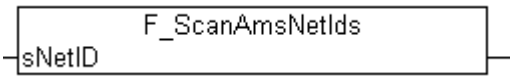
```
PROGRAM MAIN
VAR
    ids     : T_AmsNetIdArr := [127, 16, 17, 3, 1, 1];
    sNetID  : T_AmsNetID := '';
END_VAR
```

```
sNetID := F_CreateAmsNetId( ids );(* Result: '127.16.17.3.1.1' *)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.5 F_ScanAmsNetIds



The function F_ScanAmsNetIds converts a string with the TwinCAT network address into single address bytes. The single address bytes are converted from left to right. They are returned as an array of bytes. The address bytes are represented in network byte order.

FUNCTION F_ScanAmsNetIds : T_AmsNetIdArr

T_AmsNetIdArr [▶ 108]

```
VAR_INPUT
    sNetID : T_AmsNetID;
END_VAR
```

sNetID: TwinCAT network address [▶ 107] as string. E.g.: '127.16.17.3.1.1'

Input value	Return value	Description
sNetID ≠ '' (empty string) and sNetID ≠ '0.0.0.0.0.0'	All bytes are zero	Error during conversion. Please check the format of sNetId input string.

Example in structured text:

TwinCAT network address string: '127.16.17.3.1.1' is converted to an array of address bytes.

```
PROGRAM MAIN
VAR
    ids     : T_AmsNetIdArr;
    sNetID  : T_AmsNetID := '127.16.17.3.1.1';
END_VAR
```

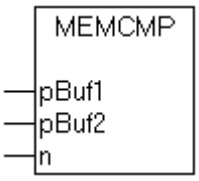
```
ids := F_ScanAmsNetIds( sNetID );
(* Result: ids[0]:=127, ids[1]:=16, ids[2]:=17, ids[3]:=3, ids[4]:=1, ids[5]:=1 *)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4 MEMORY functions

4.4.1 MEMCMP



The function MEMCMP allows the values of PLC variables in two different memory areas to be compared.

FUNCTION MEMCMP : DINT

```
VAR_INPUT
    pBuf1     :UDINT;
    pBuf2     :UDINT;
    n         :UDINT;
END_VAR
```

pBuf1: start address of the first memory area (the first data buffer).

pBuf2: start address of the second memory area (the second data buffer).

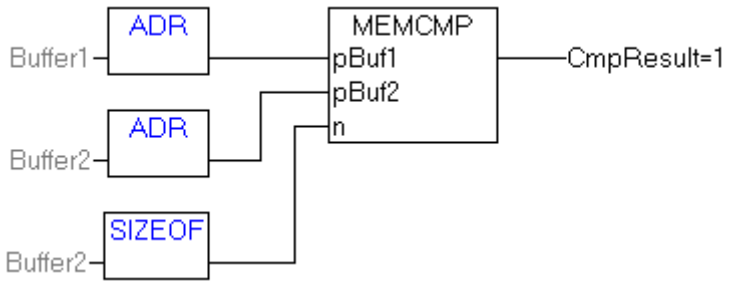
n: number of bytes to be compared.

The function compares the first n bytes in the two data buffers and returns a value that corresponds to their relationship.

Return parameter	Relationship of the first byte that differs between the first and second data buffers
-1	pBuf1 smaller than *pBuf2*
0	pBuf1 identical to *pBuf2*
1	pBuf1 greater than *pBuf2*
0xFF	Incorrect parameter values. *pBuff1* = 0 or *pBuff2* = 0 or *n* = 0

Example of a call in FBD

```
VAR
    Buffer1    : ARRAY[0..3] OF BYTE;
    Buffer2    : ARRAY[0..3] OF BYTE;
    CmpResult  : DINT;
END_VAR
```

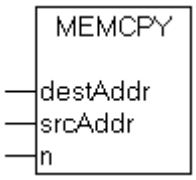


In this example, 4 bytes of data in *Buffer2* are compared with those in *Buffer1*. The first differing byte is larger in *Buffer1* than it is in *Buffer2*.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCHelper.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4.2 MEMCPY



The function MEMCPY can be used to copy the values of PLC variables from one memory area to another.

FUNCTION MEMCPY : UDINT

```
VAR_INPUT
    destAddr      : UDINT;
    srcAddr       : UDINT;
    n             : UDINT;
END_VAR
```

destAddr: start address of the target memory area.

srcAddr: start address of the source memory area.

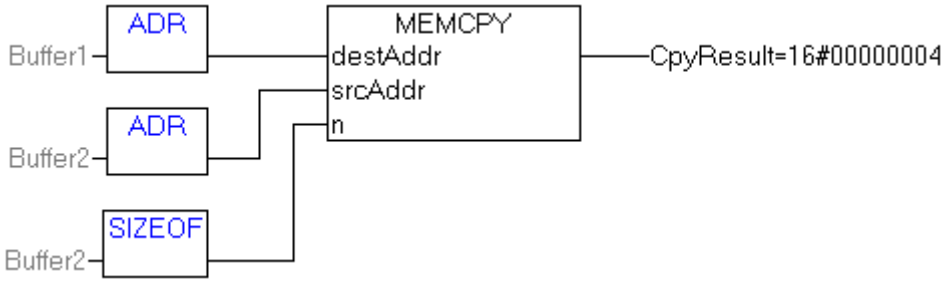
n: number of bytes to be copied.

The function copies *n* bytes from the memory area that starts at *srcAddr* to the memory area that starts at *destAddr*.

Return parameter	Meaning
0	Incorrect parameter values. *destAddr == 0* or *srcAddr==0* or *n == 0*
> 0	If successful, the number of bytes copied (*n*).

Example of a call in FBD

```
VAR
    Buffer1    : ARRAY[0..3] OF BYTE;
    Buffer2    : ARRAY[0..3] OF BYTE;
    CpyResult  : UDINT;
END_VAR
```

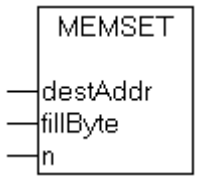


In the example, 4 bytes are copied from *Buffer2* to *Buffer1*.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCHelper.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4.3 MEMSET



The function MEMSET allows PLC variables in a particular memory area to be set to a particular value.

FUNCTION MEMSET : UDINT

```
VAR_INPUT
    destAddr  :UDINT;
    fillByte  :USINT;
    n         :UDINT;
END_VAR
```

destAddr: start address of the memory area that is to be set.

fillByte: value of the filler byte.

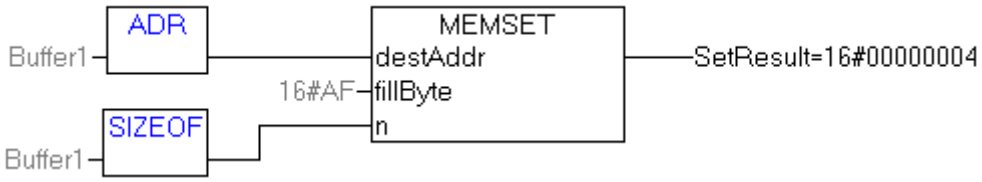
n: number of bytes to be set.

The function fills *n* bytes in the memory area that starts at address *destAddr* with the value specified by *fillByte*.

Return parameter	Meaning
0	Incorrect parameter values. *destAddr == 0* or *n == 0*
> 0	If successful, the number of bytes that have been set (*n*).

Example of a call in FBD

```
VAR
    Buffer1    : ARRAY[0..3] OF BYTE;
    SetResult  : UDINT;
END_VAR
```

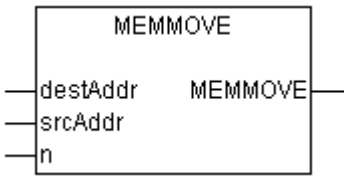


In the example, 4 bytes in *Buffer1* are set to the value 0xAF.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCHelper.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4.4 MEMMOVE



The function MEMMOVE can be used to copy the values of PLC variables from one memory area to another. If some regions of the source area and the destination overlap, MEMMOVE ensures that the original source bytes in the overlapping region are copied before being overwritten.

FUNCTION MEMMOVE : UDINT

```
VAR_INPUT
    destAddr      : UDINT;
    srcAddr       : UDINT;
    n             : UDINT;
END_VAR
```

destAddr: start address of the target memory area.

srcAddr: start address of the source memory area.

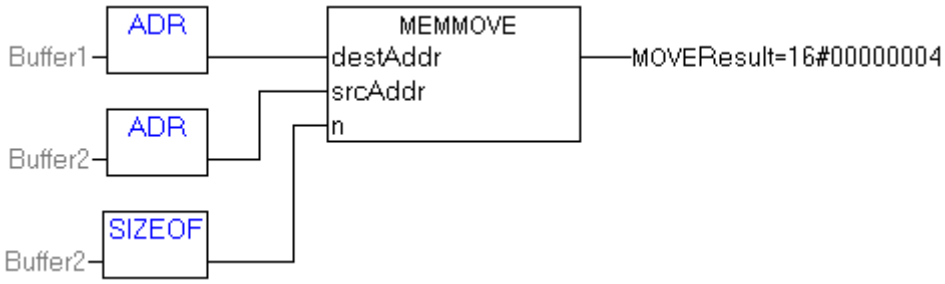
n: number of bytes to be copied.

The function copies *n* bytes from the memory area that starts at *srcAddr* to the memory area that starts at *destAddr*.

Return parameter	Meaning
0	Incorrect parameter values. *destAddr == 0* or *srcAddr==0* or *n == 0*
> 0	If successful, the number of bytes copied (*n*).

Example of a call in FBD

```
VAR
    Buffer1     : ARRAY[0..3] OF BYTE;
    Buffer2     : ARRAY[0..3] OF BYTE;
    MoveResult  : UDINT;
END_VAR
```



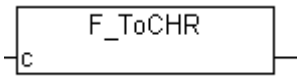
In the example, 4 bytes are moved from *Buffer2* to *Buffer1*.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 737	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.5 Character functions

4.5.1 F_ToCHR



The function converts ASCII Code zu STRING.

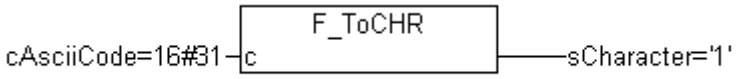
FUNCTION F_ToCHR: STRING

```
VAR_INPUT
    c       : BYTE;
END_VAR
```

c: ASCII-Code to be converted;

Example for a call in FBD:

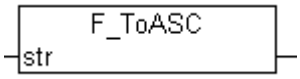
```
PROGRAM P_TEST
VAR
    sCharacter : STRING(1) := '';
    cAsciiCode : BYTE := 16#31;
END_VAR
```



Requirements

Development environment	Target System	PLC Libraries to include
TwinCAT v2.10.0 Build > 1256	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.5.2 F_ToASC



This function converts STRING to ASCII Code. Only the first sign of the STRING will be converted. An empty STRING delivers a zero.

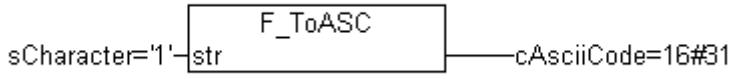
FUNCTION F_ToASC : BYTE

```
VAR_INPUT
    str : STRING;
END_VAR
```

str: STRING to be converted.

Example for a call in FBD:

```
PROGRAM P_TEST
VAR
    sCharacter : STRING(1) := '1';
    cAsciiCode : BYTE := 0;
END_VAR
```



Requirements

Development environment	Target System	PLC Libraries to include
TwinCAT v2.10.0 Build > 1256	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5 Data structures

5.1 SYSTEMINFOTYPE

```
TYPE SYSTEMINFOTYPE
STRUCT
    runTimeNo           : BYTE;
    projectName         : STRING(32);
    numberOfTasks       : BYTE;
    onlineChangeCount   : UINT;
    bootDataFlags       : BYTE;
    systemStateFlags    : WORD;
END_STRUCT
END_TYPE
```

runTimeNo : The number of the active runtime system (1..4).

projectName : The name of the PLC project.

numberOfTasks : The number of tasks (max. 4) in the actual runtime system.

onlineChangeCount : The number of online changes made since the last complete download.

bootDataFlags : The status of the boot data (RETAIN and PERSISTENT) after loading. The upper four bits indicate the state of the persistent data, while the lower four bits indicate the state of the retain data.

Bit number	Description
0	RETAIN variables: LOADED (without error)
1	RETAIN variables: INVALID (the back-up copy was loaded, since no valid data was present)
2	RETAIN variables: REQUESTED (RETAIN variables should be loaded, a setting in TwinCAT System Control)
3	reserved
4	PERSISTENT variables: LOADED (without error)
5	PERSISTENT variables: INVALID (the back-up copy was loaded, since no valid data was present)
6	reserved
7	reserved

systemStateFlags : Reserved.

When shutting TwinCAT down the PERSISTENT and RETAIN data is written into two files on the hard disk. The path can be specified in TwinCAT System Control by means of the TwinCAT system properties (PLC tab). The standard setting is "*<Drive>:\TwinCAT\Boot*". The files all have a fixed name with fixed extensions.

File name	Description
TCPLC_P_x.wbp	Boot project (x = number of the run-time system)
TCPLC_S_x.wbp	Packed Sourcecode (x = number of the run-time system)
TCPLC_R_x.wbp	RETAIN variables (x = number of the run-time system)
TCPLC_T_x.wbp	PERSISTENT variables (x = number of the run-time system)
TCPLC_R_x.wb~	Backup copy of the RETAIN variables (x = number of the run-time system)
TCPLC_T_x.wb~	Backup copy of the PERSISTENT variables (x = number of the run-time system)

If the file for the persistent and/or retain variables can not be written when shutting TwinCAT down, the standard reaction is for the backup file to be loaded. In that case bit 1 of the bootDataFlags (for the RETAIN variables) in the PLC and/or bit 5 (for the PERSISTENT variables) is set.

If the back-up file is not to be used under any conditions, a setting must be made in the NT registry. In the registry editor, under

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\Plc]

"ClearInvalidRetainData"=dword:00000000

"ClearInvalidPersistentData"=dword:00000000
```

the value of "ClearInvalidRetainData" or of "ClearInvalidPersistentData" must be set to 1. The default setting is 0.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.2 SYSTEMTASKINFOTYPE

```
TYPE SYSTEMTASKINFOTYPE
STRUCT
    active             : BOOL;
    taskName           : STRING(16);
    firstCycle         : BOOL;
    cycleTimeExceeded  : BOOL;
    cycleTime          : UDINT;
    lastExecTime       : UDINT;
    priority           : BYTE;
    cycleCount         : UDINT;
END_STRUCT
END_TYPE
```

active : Boolean variable is TRUE, if the task is active.

taskName : Contains the name of the task.

firstCycle : The variable is TRUE in the first cycle of the task.

cycleTimeExceeded : The variable is TRUE if the runtime of the task exceeds the cycletime set in taskconfiguration.

cycleTime : Cycletime set in the taskconfiguration in parts of 100ns.

lastExecTime : Needed cycletime for the last cycle in parts of 100ns.

priority : Priority set in the task configuration.

cycleCount : Cyclecounter.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.3 T_AmsNetId

```
TYPE T_AmsNetId : STRING(23);
END_TYPE
```

The variable of this type is a string containing the AMS network identifier of the target device to which the ADS command is directed. The string consists of six numerical fields, separated by dots. Each numerical field contains a number between 1 and 254. Valid AMS network addresses are, for example, "1.1.1.2.7.1" or "200.5.7.170.1.7". If an empty string is passed, the AMS network identifier of the local device is automatically assumed.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.4 T_AmsNetIdArr

```
TYPE T_AmsNetIdArr : ARRAY[0..5] OF BYTE;
END_TYPE
```

The variable of this type is a array of bytes containing the AMS network identifier. The address bytes are represented in network byte order.
E.g. '127.16.17.3.1.1' is represented as:
byte[0] = 127
byte[1] = 16
byte[2] = 17
byte[3] = 3
byte[4] = 1
byte[5] = 1

Example: F_ScanAmsNetIds [▶ 98]

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86, ARM)	TcSystem.Lib

5.5 T_AmsPort

```
TYPE T_AmsPort : UINT;
END_TYPE
```

ADS devices in the TwinCAT network are identified by an AMS network address and a port number. The following port numbers are invariably specified on every individual TwinCAT system. Table with specified ADS port numbers:

ADS device	Port number
Camshaft controller value	900
PLC	Run-time system 1 :801 Run-time system 2 :811 Run-time system 3 :821 Run-time system 4 :831
NC	500
Reserved	400
I/O	300
Real-time kernel	200
Message system	100

Up to 4 independent PLC run-time systems can run on each computer, and each run-time system is to be looked on as an independent PLC. The port number and the network address are both required as input parameters when the ADS blocks are called.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.6 T_MaxString

```
TYPE T_MaxString : STRING(MAX_STRING_LENGTH);
END_TYPE
```

The variable of this type is PLC string with the maximal length.

```
VAR_GLOBAL CONSTANT
     MAX_STRING_LENGTH : UDINT := 255;
ENd_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.7 T_IPv4Addr

```
TYPE T_IPv4Addr: STRING(15);
END_TYPE
```

String containing an (Ipv4) Internet Protocol dotted address. E.g. '172.16.7.199'

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcSystem.Lib

5.8 T_IPv4AddrArr

```
TYPE T_IPv4AddrArr: ARRAY[0..3] OF BYTE;
END_TYPE
```

The variable of this type is a array of bytes containing the (IPv4) Internet Protocol network address. The address bytes are represented in network byte order.
E.g. '172.16.7.199' is represented as:

byte[0] := 172
byte[1] := 16
byte[2] := 7
byte[3] := 199

Example: F_ScanIPv4AddrIds [▶ 83]

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1340	PC or CX (x86, ARM)	TcSystem.Lib

5.9 ST_AmsAddr

```
TYPE ST_AmsAddr :
STRUCT
    netId   : T_AmsNetIdArr;
    port    : T_AmsPort;
END_STRUCT
END_TYPE
```

netId: Ams network ID address bytes [▶ 108];

port: Ams port number [▶ 108];

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86, ARM)	TcSystem.Lib

5.10 E_OpenPath

```
TYPE E_OpenPath :
(
    PATH_GENERIC    :=1,    (* search/open/create files in selected/generic folder *)
    PATH_BOOTPRJ,       (* search/open/create files in the TwinCAT/
Boot directory (adds the extension .wbp) *)
    PATH_BOOTDATA,      (* reserved for future use*)
    PATH_BOOTPATH,      (* refers to the TwinCAT/
Boot directory without adding an extension (.wbp) *)
    PATH_USERPATH1  :=11,   (*reserved for future use*)
    PATH_USERPATH2,     (*reserved for future use*)
    PATH_USERPATH3,     (*reserved for future use*)
    PATH_USERPATH4,     (*reserved for future use*)
    PATH_USERPATH5,     (*reserved for future use*)
    PATH_USERPATH6,     (*reserved for future use*)
    PATH_USERPATH7,     (*reserved for future use*)
    PATH_USERPATH8,     (*reserved for future use*)
    PATH_USERPATH9      (*reserved for future use*)
);
END_TYPE
```

The variable of this type selects generic or one of the TwinCAT system paths on the target device to perform the file open operation.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.11 E_SeekOrigin

```
TYPE E_SeekOrigin :
(
    SEEK_SET    :=      0,      (* Seek from beginning of file *)
    SEEK_CUR,               (* Seek from current position of file pointer *)
    SEEK_END            (* Seek from the end of file *)
);
END_TYPE
```

Value	Description
SEEK_SET	Seek from beginning of file
SEEK_CUR	Seek from current position of file pointer
SEEK_END	Seek from the end of file

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.12 E_TcEventClass

```
TYPE E_TcEventClass :
(
    TCEVENTCLASS_NONE               :=0,    (* No class *)
    TCEVENTCLASS_MAINTENANCE        :=1,    (* Maintenance hint *)
    TCEVENTCLASS_MESSAGE            :=2,    (* Message *)
    TCEVENTCLASS_HINT               :=3,    (* Hint *)
    TCEVENTCLASS_STATEINFO          :=4,    (* State information *)
    TCEVENTCLASS_INSTRUCTION        :=5,    (* Instruction *)
    TCEVENTCLASS_WARNING            :=6,    (* Warning *)
    TCEVENTCLASS_ALARM              :=7,    (* Alarm *)
    TCEVENTCLASS_PARAMERROR         :=8     (* Parameter error *)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.13 E_TcEventClearModes

```
TYPE E_TcEventClearModes :
(
    TCEVENTLOGIOFFS_CLEARACTIVE     := 1,
    TCEVENTLOGIOFFS_CLEARLOGGED     ,
    TCEVENTLOGIOFFS_CLEARALL
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.14 E_TcEventPriority

```
TYPE E_TcEventPriority :
(
    TCEVENTPRIO_IMPLICIT    :=      0
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.15 E_TcEventStreamType

```
TYPE E_TcEventStreamType :
(
    TCEVENTSTREAM_INVALID       := 0,   (* no source name, no prog id *)
    TCEVENTSTREAM_SIMPLE,           (* no source name, no prog id *)
    TCEVENTSTREAM_NORMAL,           (* source name AND prog id *)
    TCEVENTSTREAM_NOSOURCE,         (* no source name, but prog id *)
    TCEVENTSTREAM_CLASSID,          (* source name AND class id *)
    TCEVENTSTREAM_CLSNOSRC,         (* no source name but class id *)
    TCEVENTSTREAM_READCLASSCOUNT,   (* *)
    TCEVENTSTREAM_MAXTYPE           (* no source name but class id *)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.16 E_IOAccessSize

```
TYPE E_IOAccessSize :
(
    NoOfByte_Byte   := 1,
    NoOfByte_Word   := 2,
    NoOfByte_DWord  := 4
);
END_TYPE
```

Byte size of I/O position (Number of bytes to be written or read).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.0.10 Build > 1257	PC or CX (x86)	TcSystem.Lib

5.17 TcEvent

```
TYPE TcEvent
STRUCT
    Class                : UDINT;
    Prio                 : UDINT;
    Id                   : UDINT;
    bQuitRequired        : BOOL;
    DataFormatStrAddress : UDINT;
    UserFlags            : DWORD;
    Flags                : DWORD;
    StreamType           : UDINT;
    SourceString         : STRING[15];     ( TCEVENT_SRCNAMESIZE )
    SourceId             : UDINT;
    ProgId               : STRING[31];     ( TCEVENT_FMTPRGSIZE )
END_STRUCT
END_TYPE
```

Class : Event class, take value from the enum E_TcEventClass [▶ 111]

Prio : priority of the event inside a class, free selectable count (1..MaxUDINT)

Id : Id of the events, is used for explicit identification in the Eventlogger

bQuitRequired : edges for switch on and off the acknowledgement obligatory (TRUE --> acknowledgement obligatory)

DataFormatStrAddress : Address of a strings. String contains formatting instructions (e.g. '%d%f' formates an Integer and a Real (float) value)

UserFlags : 32-bit count is free available

Flags : 32-bit count for identification of the event. The meaning of the single bits is declared in the global Variables [▶ 115] of the libraries

StreamType : Type of events. Take the value from the enum E_TcEventStreamType [▶ 112]

SourceString : String with the source name (max. 15 strings [▶ 115]).

SourceId : Source-ID.

ProgId : String (Prog-Id) with the name of the formatter (max. 31 strings [▶ 115])

TwinCAT 2.7 default: 'TcEventLogger.TcLogFormatter'

TwinCAT 2.8 default: 'TcEventLogger.TcLogFormatter' or 'TcEventFormatter.TcXmlFormatter'

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.18 PVOID

```
TYPE PVOID : UDINT;
END_TYPE
```

Platform independent pointer type:

- x86, ARM => 32 bit pointer type (TwinCAT 2.xx default);
- x64 => 64 bit pointer type (reserved for future use in TwinCAT 3.x);

Requirements

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

5.19 UXINT

```
TYPE UXINT : UDINT;
END_TYPE
```

Platform independent unsigned integer type:

- x86, ARM => 32 bit unsigned integer (TwinCAT 2.xx default);
- x64 => 64 bit unsigned integer (reserved for future use in TwinCAT 3.x);

Requirements

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

5.20 XINT

```
TYPE XINT : DINT;
END_TYPE
```

Platform independent signed integer type:

- x86, ARM => 32 bit signed integer (TwinCAT 2.xx default);
- x64 => 64 bit signed integer (reserved for future use in TwinCAT 3.x);

Requirements

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

5.21 XWORD

```
TYPE XWORD : DWORD;
END_TYPE
```

Platform independent bit type:

- x86, ARM => 32 bit unsigned integer (TwinCAT 2.xx default);
- x64 => 64 bit unsigned integer (reserved for future use in TwinCAT 3.x);

Requirements

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

6 Constants

In the TcSystem.lib different constants were defined.

Port numbers	Value	Description
AMSPORT_LOGGER	100	Port number of the standard loggers.
AMSPORT_EVENTLOG	110	Port number of the TwinCAT Eventloggers.
AMSPORT_R0_RTIME	200	Port number of the TwinCAT Realtime Servers.
AMSPORT_R0_IO	300	Port number of the TwinCAT I/O Servers.
AMSPORT_R0_NC	500	Port number of the TwinCAT NC Servers.
AMSPORT_R0_NCSAF	501	Port number of the TwinCAT NC Servers (Task SAF).
AMSPORT_R0_NCSVB	511	Port number of the TwinCAT NC Servers (Task SVB).
AMSPORT_R0_ISG	550	internal
AMSPORT_R0_CNC	600	Port number of the TwinCAT NC I Servers.
AMSPORT_R0_LINE	700	internal
AMSPORT_R0_PLC	800	Port number of the TwinCAT PLC Servers (only at the Buscontroller).
AMSPORT_R0_PLC_RTS1	801	Port number of the TwinCAT PLC Servers in the runtime 1.
AMSPORT_R0_PLC_RTS2	811	Port number of the TwinCAT PLC Servers in the runtime 2.
AMSPORT_R0_PLC_RTS3	821	Port number of the TwinCAT PLC Servers in the runtime 3.
AMSPORT_R0_PLC_RTS4	831	Port number of the TwinCAT PLC Servers in the runtime 4.
AMSPORT_R0_CAM	900	Port number of the TwinCAT CAM Server.
AMSPORT_R0_CAMTOOL	950	Port number of the TwinCAT CAMTOOL Server.
AMSPORT_R3_SYSSERV	10000	Port number of the TwinCAT System Service.
AMSPORT_R3_SCOPESERVER	27110	Port number of the TwinCAT Scope Servers (since Lib. V2.0.12)

ADS States	Value	Description
ADSSTATE_INVALID	0	ADS Status: invalid
ADSSTATE_IDLE	1	ADS Status: idle
ADSSTATE_RESET	2	ADS Status: reset.
ADSSTATE_INIT	3	ADS Status: init
ADSSTATE_START	4	ADS Status: start
ADSSTATE_RUN	5	ADS Status: run
ADSSTATE_STOP	6	ADS Status: stop
ADSSTATE_SAVECFG	7	ADS Status: save configuration
ADSSTATE_LOADCFG	8	ADS Status: load configuration
ADSSTATE_POWERFAILURE	9	ADS Status: Power failure
ADSSTATE_POWERGOOD	10	ADS Status: Power good
ADSSTATE_ERROR	11	ADS Status: Error
ADSSTATE_SHUTDOWN	12	ADS Status: Shutdown
ADSSTATE_SUSPEND	13	ADS Status: Suspend
ADSSTATE_RESUME	14	ADS Status: Resume
ADSSTATE_CONFIG	15	ADS Status: Configuration
ADSSTATE_RECONFIG	16	ADS Status: Reconfiguration
ADSSTATE_MAXSTATES	17	

Reserved Index Groups	Value	Description
ADSIGRP_SYMTAB	16#F000	
ADSIGRP_SYMNAME	16#F001	
ADSIGRP_SYMVAL	16#F002	
ADSIGRP_SYM_HNDBYNAME	16#F003	
ADSIGRP_SYM_VALBYNAME	16#F004	
ADSIGRP_SYM_VALBYHND	16#F005	
ADSIGRP_SYM_RELEASEHND	16#F006	
ADSIGRP_SYM_INFOBYNAME	16#F007	
ADSIGRP_SYM_VERSION	16#F008	
ADSIGRP_SYM_INFOBYNAMEEX	16#F009	
ADSIGRP_SYM_DOWNLOAD	16#F00A	
ADSIGRP_SYM_UPLOAD	16#F00B	
ADSIGRP_SYM_UPLOADINFO	16#F00C	
ADSIGRP_SYMNOTE	16#F010	
ADSIGRP_IOIMAGE_RWIB	16#F020	
ADSIGRP_IOIMAGE_RWIX	16#F021	
ADSIGRP_IOIMAGE_RISIZE	16#F025	
ADSIGRP_IOIMAGE_RWOB	16#F030	
ADSIGRP_IOIMAGE_RWOX	16#F031	
ADSIGRP_IOIMAGE_RWOSIZE	16#F035	
ADSIGRP_IOIMAGE_CLEARI	16#F040	
ADSIGRP_IOIMAGE_CLEARO	16#F050	
ADSIGRP_IOIMAGE_RWIOB	16#F060	
ADSIGRP_DEVICE_DATA	16#F100	
ADSIOFFS_DEVDATA_ADSSTATE	16#0000	
ADSIOFFS_DEVDATA_DEVSTATE	16#0002	

System Service Index Groups	Value	Description
SYSTEMSERVICE_OPENCREATE	100	
SYSTEMSERVICE_OPENREAD	101	
SYSTEMSERVICE_OPENWRITE	102	
SYSTEMSERVICE_CREATEFILE	110	
SYSTEMSERVICE_CLOSEHANDLE	111	
SYSTEMSERVICE_FOPEN	120	
SYSTEMSERVICE_FCLOSE	121	
SYSTEMSERVICE_FREAD	122	
SYSTEMSERVICE_FWRITE	123	
SYSTEMSERVICE_FSEEK	124	
SYSTEMSERVICE_FTELL	125	
SYSTEMSERVICE_FGETS	126	
SYSTEMSERVICE_FPUTS	127	
SYSTEMSERVICE_FSCANF	128	
SYSTEMSERVICE_FPRINTF	129	
SYSTEMSERVICE_FEOF	130	

System Service Index Groups	Value	Description
SYSTEMSERVICE_FDELETE	131	
SYSTEMSERVICE_FRENAME	132	
SYSTEMSERVICE_REG_HKEYLOCALMACHINE	200	
SYSTEMSERVICE_SENDEMAIL	300	
SYSTEMSERVICE_TIMESERVICES	400	
SYSTEMSERVICE_STARTPROCESS	500	
SYSTEMSERVICE_CHANGENETID	600	

System Service Index Offsets (Timeservices)	Value	Description
TIMESERVICE_DATEANDTIME	1	
TIMESERVICE_SYSTEMTIMES	2	
TIMESERVICE_RTCTIMEDIFF	3	
TIMESERVICE_ADJUSTTIMETORTC	4	

Masks for Log output	Value	Description
ADSLOG_MSGTYPE_HINT	16#01	
ADSLOG_MSGTYPE_WARN	16#02	
ADSLOG_MSGTYPE_ERROR	16#04	
ADSLOG_MSGTYPE_LOG	16#10	
ADSLOG_MSGTYPE_MSGBOX	16#20	
ADSLOG_MSGTYPE_RESOURCE	16#40	
ADSLOG_MSGTYPE_STRING	16#80	

Masks for Bootdata-Flags	Value	Description
BOOTDATAFLAGS_RETAIN_LOADED	16#01	
BOOTDATAFLAGS_RETAIN_INVALID	16#02	
BOOTDATAFLAGS_RETAIN_REQUESTED	16#04	
BOOTDATAFLAGS_PERSISTENT_LOADED	16#10	
BOOTDATAFLAGS_PERSISTENT_INVALID	16#20	

Masks for BSOD-Flags	Value	Description
SYSTEMSTATEFLAGS_BSOD	16#01	BSOD: Blue Screen of Death

Masks for BSOD-Flags	Value	Description
SYSTEMSTATEFLAGS_RTVIOLATION	16#02	Realtime violation, latency time overrun

Masks for File output	Value	Description
FOPEN_MODEREAD	16#0001	'r': Opens file for reading
FOPEN_MODEWRITE	16#0002	'w': Opens file for writing, (possible) existing files were overwritten.
FOPEN_MODEAPPEND	16#0004	'a': Opens file for writing, is attached to (possible) exisiting files. If no file exists, it will be created.
FOPEN_MODEPLUS	16#0008	'+': Opens a file for reading and writing.
FOPEN_MODEBINARY	16#0010	'b': Opens a file for binary reading and writing.
FOPEN_MODETEXT	16#0020	't': Opens a file for textual reading and writing.

Masks for Eventlogger Flags	Value	Description
TCEVENTFLAG_PRIOCLASS	16#0010	Class and priority are defined by the formatter.
TCEVENTFLAG_FMTSELF	16#0020	The formatting information comes with the event
TCEVENTFLAG_LOG	16#0040	Logg.
TCEVENTFLAG_MSGBOX	16#0080	Show message box .
TCEVENTFLAG_SRCID	16#0100	Use Source-Id instead of Source name.

TwinCAT Eventlogger Status messages	Value	Description
TCEVENTSTATE_INVALID	16#0000	Not valid, occurs also if the event was not reported.
TCEVENTSTATE_SIGNALED	16#0001	Event is reported, but neither signed off nor acknowledged.
TCEVENTSTATE_RESET	16#0002	Event is signed off ('gone').
TCEVENTSTATE_CONFIRMED	16#0010	Event is acknowledged.
TCEVENTSTATE_RESETCON	16#0012	Event is signed off and acknowledged.

TwinCAT Eventlogger Status messages	Value	Description
TCEVENT_SRCNAMESIZE	15	Max. Length for the Source name.
TCEVENT_FMTPRGSIZE	31	Max. Length for the name of the formatters.

Other	Value	Description
PI	3.14159265358979323846264338 32795	Pi number
DEFAULT_ADS_TIMEOUT	T#5s	Default ADS timeout
MAX_STRING_LENGTH	255	The max. string length of T_MaxString data type

More Information:
www.beckhoff.com/tx1200

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com