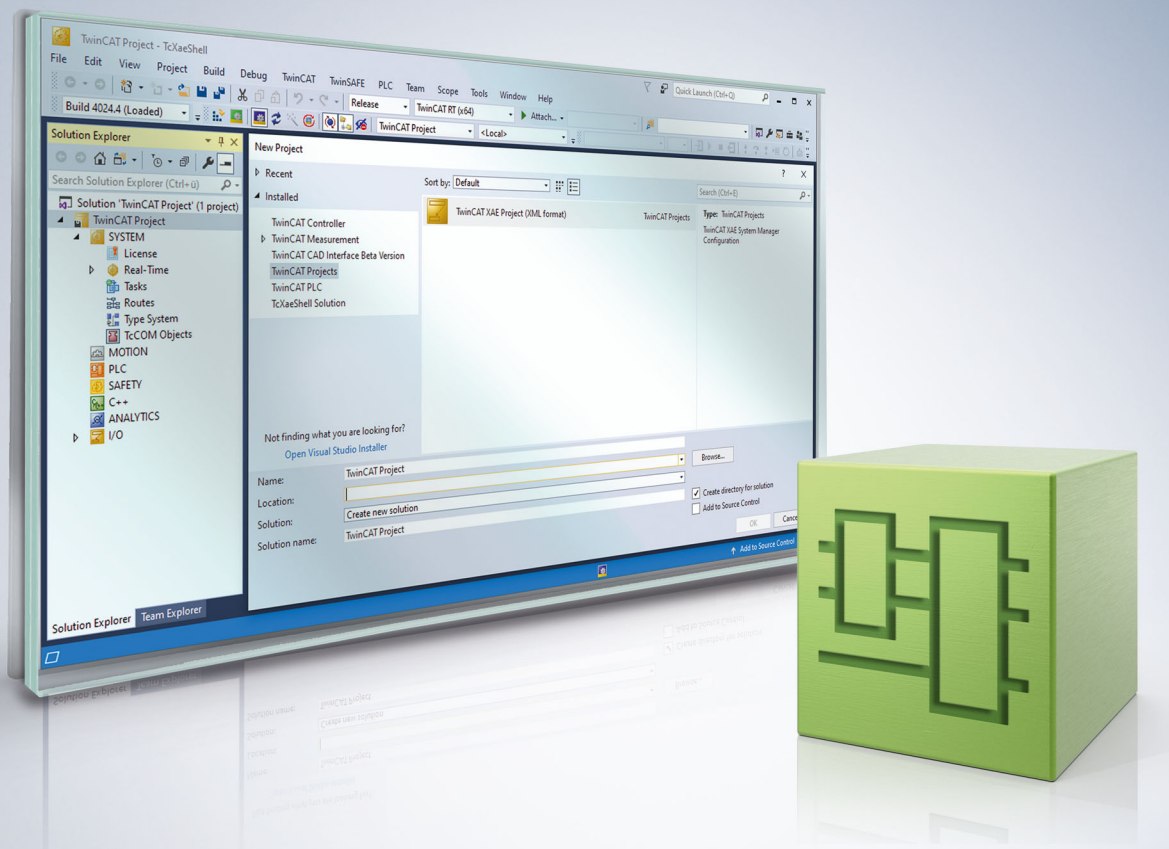
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC Library: Tc2_System



2023-12-19 | Version: 1.16.2

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all 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 notice.
No claims to modify 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 and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

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

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

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

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 here.

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.

2 Overview

Not all those function blocks and functions that are often needed in PLC applications are standardized in IEC61131-3. The Tc2_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.

General function blocks

Name	Description
DRAND [▶ 14]	Random number generator
FB_IecCriticalSection [▶ 14]	Critical sections should be made mutually exclusive
FB_SetLedColor_BAPI [▶ 17]	Switches the user LED to PCs and Émbedded PCs with BIOS API support
GETCURTASKINDEX [▶ 19]	Determines the index of the current task

ADS function blocks

Name	Description
ADSREAD [▶ 39]	Reading data via ADS
ADSREADEX [▶ 40]	Reading data via ADS and returning the number of read data bytes
ADSWRITE [▶ 41]	Writing data via ADS
ADSRDWRT [▶ 43]	Reading and writing data via ADS
ADSRDWRTEX [▶ 45]	Writing data via ADS and returning the number of read data bytes
ADSRDSTATE [▶ 21]	Read the state of a device via ADS
ADSWRTCTL [▶ 23]	Write control words to a device via ADS
ADSRDDEVINFO [▶ 24]	Read device information via ADS

Expanded ADS function blocks

Name	Description
ADSREADIND [▶ 27]	ADSREAD Indication
ADSWRITEIND [▶ 30]	ADSWRITE Indication
ADSRDWRTIND [▶ 33]	ADSRDWRT Indication
ADSREADRES [▶ 36]	ADSREAD Response
ADSWRITERES [▶ 37]	ADSWRITE Response
ADSRDWRTRES [▶ 38]	ADSRDWRT Response

Function 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_EOF [▶ 47]	Check the end of file
FB_FileOpen [▶ 48]	Open a file
FB_FileClose [▶ 51]	Close a file
FB_FileGets [▶ 53]	Get string from a file
FB_FilePuts [▶ 55]	Put string to a file
FB_FileRead [▶ 56]	Read from a file
FB_FileWrite [▶ 58]	Write to a file
FB_FileSeek [▶ 60]	Move the file pointer
FB_FileTell [▶ 61]	Get the file pointer position
FB_FileDelete [▶ 62]	Delete a file
FB_FileRename [▶ 64]	Rename a file
FB_CreateDir [▶ 65]	Create new directory
FB_RemoveDir [▶ 67]	Remove directory

EventLogger function blocks

The TwinCAT EventLogger has the task of managing all messages (events) appearing in the TwinCAT system; to forward them and where necessary to write them into the TwinCAT log file.

Name	Description
ADSLOGEVENT [▶ 68]	Sending and acknowledging TwinCAT EventLogger messages.
ADSCLEAREVENTS [▶ 71]	Sending and acknowledging TwinCAT EventLogger messages.
FB_SimpleAdsLogEvent [▶ 72]	Sending and acknowledging TwinCAT EventLogger messages.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

The TwinCAT EventLogger was replaced by the successor TwinCAT 3 EventLogger. The older TwinCAT EventLogger is supported by TwinCAT 3 up to version 3.1.4024. Newer TwinCAT versions (>= 3.1.4026.0) only support the newer TwinCAT 3 EventLogger. PLC function blocks for this can be found in the PLC library Tc3_EventLogger.

IEC steps / SFC flags function blocks

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

Name	Description
AnalyzeExpression [▶ 74]	Required if SFC flags are used
AnalyzeEspressionTable [▶ 76]	Required if SFC flags are used
AnalyzeExpressionCombined [▶ 79]	Required if SFC flags are used
AppendErrorString [▶ 79]	Required if SFC flags are used in order to format strings with error description
SFCActionControl [▶ 79]	Enables the use of IEC steps

Watchdog function blocks

Name	Description
FB_PcWatchdog [▶ 80]	Activates or deactivates the PC watchdog Only available on IPCs with the following mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051
FB_PcWatchdog_BAPI [▶ 81]	Activates or deactivates the PC watchdog Only available on IPCs with the following mainboards: CBxx63 with a BIOS version >=0.44

Time function blocks

Name	Description
GETCPUCOUNTER [▶ 83]	Read the CPU cycle counter
GETCPUACCOUNT [▶ 83]	Read the PLC task cycle counter

General functions

Name	Description
F_CheckMemoryArea [▶ 85]	Returns information about the memory area in which the requested variable with the specified size is located
F_CmpLibVersion [▶ 86]	Compares an existing library with the required version
F_CreateIPv4Addr [▶ 86]	Converts individual IPv4 address bytes to a string
F_ScanIPv4AddrIds [▶ 87]	Converts an IPv4 address string to individual address bytes
F_GetMappingPartner [▶ 89]	Returns the object ID of the partner side of the mapping
F_GetMappingStatus [▶ 89]	Returns the current mapping status of a PLC variable
F_GetStructMemberAlignment [▶ 90]	Reads information about the memory alignment used
F_SplitPathName [▶ 94]	Splits the path name into four individual components
SETBIT32 [▶ 95]	Sets a bit in a DWORD
CSETBIT32 [▶ 96]	Setting/resetting of a bit in a DWORD
GETBIT32 [▶ 96]	Reads a bit from a DWORD
CLEARBIT32 [▶ 97]	Clears a bit in a DWORD
GETCURTASTINDEXEX [▶ 98]	Determining the task index
LPTSIGNAL [▶ 98]	Outputs a signal on one of the parallel port pins
TestAndSet [▶ 99]	Setting and checking a flag without option to interrupt it

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 [▶ 100]	Log a DINT variable into NT Eventlog and/or Messagebox
ADSLOGLREAL [▶ 102]	Log a (L)REAL variable into NT Eventlog and/or Messagebox
ADSLOGSTR [▶ 103]	Log a STRING variable into NT Eventlog and/or Messagebox
F_CreateAmsNetId [▶ 105]	Creates AmsNetId string
F_ScanAmsNetIds [▶ 105]	Converts AmsNetId string to array of address bytes

Character functions

Name	Description
F_ToASC [▶ 107]	Converts string character to ASCII number
F_ToCHR [▶ 106]	Converts ASCII number to string character

I/O port access

Name	Description
F_IOPortRead [▶ 107]	Reads from I/O Port
F_IOPortWrite [▶ 108]	Writes to I/O Port

Memory functions

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

NOTICE

System crash or access to forbidden memory areas

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 [▶ 110]	Compares the values of variables in two memory areas
MEMCPY [▶ 111]	Copies the values of variables from one memory area to another
MEMMOVE [▶ 112]	Copies the values from overlapping memory areas
MEMSET [▶ 113]	Sets the variables in a memory area to a particular value

Time functions

Name	Description
F_GetSystemTime [▶ 114]	Read the operating system time stamp
F_GetTaskTime [▶ 115]	Read the target start time of the task

3 Function blocks

3.1 General function blocks

3.1.1 DRAND

The function block permits generation of a (pseudo-) random number of type LREAL.

Inputs

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

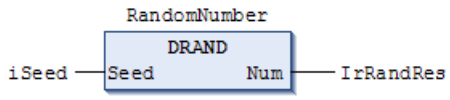
Name	Type	Description
Seed	INT	Initial value for specification of the random number series.

Outputs

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

Name	Type	Description
Num	LREAL	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 the function 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.1.2 FB_IecCriticalSection

The function block is used to make critical sections mutually exclusive. Critical sections are characterized by modifications affecting one or usually several variables, which have an inconsistent state during modifications. It is therefore imperative that such modifications are only carried out by one task at a time. The function block provides the methods Enter() and Leave() for this purpose. A successful call of Enter() makes the critical section accessible; the section is then regarded as occupied. Once the modifications are complete, the critical section must be exited through Leave().

Cycle timeout due to stopped task

If another task tries to access an occupied critical area through an Enter() call, it is blocked by the TwinCAT scheduler. The task is blocked until the section is enabled again! Once enabled, processing of the program code continues, and the critical section is entered.

- Ensure that the critical sections are kept short, in order to avoid cycle overruns in the waiting task. If several tasks are waiting to enter the critical section, access is granted based on their priority.

If a task is blocked by the TwinCAT scheduler because it attempted to enter an occupied critical area, this is done without "busy waiting". Low-priority tasks can therefore utilize the CPU capacity during this time.

Windows CE

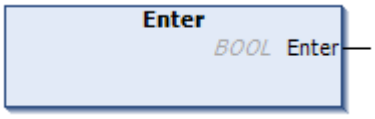
The functionality is supported under Windows CE operating systems from TwinCAT v3.1.4022.29 onwards. (In older TwinCAT versions the methods return FALSE.)

Alternative

Critical sections can also be realized with the function TestAndSet() [▶ 99]. The function can be used to select and check the content of a critical section. However, the function does not have a blocking effect, and it is possible that the section cannot be processed in a cycle.

As a rule, the number and length of the critical sections should be kept as small as possible.

Enter() method



The method marks the start of a critical section.

Possible return values:

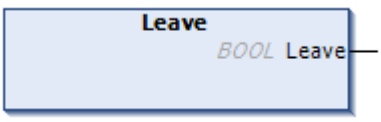
TRUE:

- The critical section may be entered.

FALSE:

- The critical section may not be entered.
- The function block is not yet supported by the runtime.
- The critical section is occupied by another PLC task. This task is on stop in a break point. The return value FALSE avoids permanent blocking of the task and ensures updating of the I/O.

Leave() method



The method marks the end of a critical section. It must always be called when a critical section is completed.

Possible return values:

TRUE:

- The section was exited successfully.

FALSE:

- The function block is not supported by the runtime.
- The critical section was not occupied with Enter.

Application sample for the function block:

The function block FB_IecCriticalSection enables access to shared files to be secured. The instance of the function block and the data to be secured are created globally.

```
VAR_GLOBAL
    fbCrititcalSection : FB_IecCriticalSection;
END_VAR

IF fbCrititcalSection.Enter() THEN
    (* start of critical section *)

    (* end of critical section *)
    fbCrititcalSection.Leave();
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.4020	PC or CX (x86, x64) WES, WES7, Win7, Win10	Tc2_System (system)
TwinCAT v3.1.4022.29	PC or CX (x86, ARM) WinCE	Tc2_System (system)

3.1.3 FB_ReadTaskExceedCounter

The function block reads the Exceed Counter. The Exceed Counter is incremented by the system whenever the selected task exceeds the set task time. This means that the real-time could not be maintained in the cycle.

The reasons for exceeding the real-time can be very diverse, but usually it is due to the PLC runtime and the application within this runtime. An example of this are programming loops such as FOR, WHILE, REPEAT, since these are always processed in one cycle.

Inputs

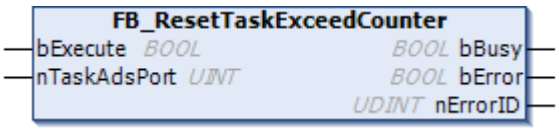
Name	Type	Description
bExecute	BOOL	A positive edge activates the function block.
nTaskAdsPort	UINT	ADS Port of the selected task Example of a possible assignment: TwinCAT_SystemInfoVarList._TaskInfo[GETCURTASKINDEX()].AdsPort

Outputs

Name	Type	Description
bBusy	BOOL	The function block is active and working.
bError	BOOL	The function block has an error.
nErrorID	UDINT	ADS error code
nExceedCounter	UDINT	Read value of the Exceed Counter

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.22	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.25.0

3.1.4 FB_ResetTaskExceedCounter



The function block can reset the Exceed Counter. The Exceed Counter is incremented whenever the selected task exceeds the set task time. This means that the real-time could not be maintained in the cycle.

The reasons for exceeding the real-time can be very diverse, but usually it is due to the PLC runtime and the application within this runtime. An example of this are programming loops such as FOR, WHILE, REPEAT, since these are always processed in one cycle.

Inputs

Name	Type	Description
bExecute	BOOL	A positive edge activates the function block.
nTaskAdsPort	UINT	ADS Port of the selected task. Example of a possible assignment: TwinCAT_SystemInfoVarList._TaskInfo[GETCURTASKINDEXEX()].AdsPort

Outputs

Name	Type	Description
bBusy	BOOL	The function block is active and working.
bError	BOOL	The function block has an error.
nErrorID	UDINT	ADS error code

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.22	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.25.0

3.1.5 FB_SetLedColor_BAPI

This functionality is only available on IPCs and Embedded PCs with a Usr-LED and with a BIOS version which supports the BIOS-API.

The function block FB_SetLedColor_BAPI can be used to switch the user LED to PCs and embedded PCs with BIOS API support. The LED color is switched via a positive edge at bExecute and the eNewColor parameter. The LED can be switched off (eNewColor = eUsrLED_Off) or set to red (eNewColor = eUsrLED_Red), blue (eNewColor = eUsrLED_Blue) or green (eNewColor = eUsrLED_Green).

Inputs

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    eNewColor       : E_UsrLED_Color;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the device (empty string or local network ID) (type T_AmsNetID [▶ 122])
eNewColor	E_UsrLED_Color	New LED color (type E_UsrLED_Color [▶ 120])
bExecute	BOOL	The command is executed with a rising edge. The input must be reset as soon as the function block is no longer active (bBusy=FALSE).
tTimeout	TIME	Time until the internal ADS communication is aborted.

Outputs

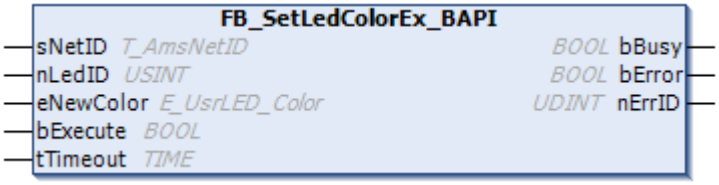
```
VAR_OUTPUT
    bBusy    : BOOL;
    bError   : BOOL;
    nErrID   : UDINT;
END_VAR
```

Name	Type	Description
bBusy	BOOL	TRUE, as long as the function block is active.
bError	BOOL	TRUE if an error occurs during command execution.
nErrID	UDINT	Contains the ADS error code or the command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System) v3.4.14

3.1.6 FB_SetLedColorEx_BAPI



This functionality is only available on IPCs and Embedded PCs with a Usr-LED and with a BIOS version which supports the BIOS-API.

The function block FB_SetLedColorEx_BAPI can be used to switch the user LEDs (USR, U1 or U2) on PCs and embedded PCs with BIOS API support. The LED color is switched via a rising edge at bExecute and the eNewColor parameter. The LED can be switched off (eNewColor = eUsrLED_Off) or set to red (eNewColor = eUsrLED_Red), blue (eNewColor = eUsrLED_Blue) or green (eNewColor = eUsrLED_Green).

Inputs

```
VAR_INPUT
    sNetID          : T_AmsNetID;
    nLedID          : USINT
    eNewColor       : E_UsrLED_Color;
    bExecute        : BOOL;
    tTimeout        : TIME;
END_VAR
```

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the device (empty string or local network ID) (type T_AmsNetID [▶ 122])
nLedID	USINT	ID for the selection of the user LED: For devices with only one USR LED, the USR LED is selected via nLedID = 0 (default value is 0). For devices with several user LEDs, the U1 LED is selected via nLedID = 1 or the U2 LED via nLedID = 2.
eNewColor	E_UsrLED_Color	New LED color (type E_UsrLED_Color [▶ 120])
bExecute	BOOL	The command is executed with a rising edge. The input must be reset as soon as the function block is no longer active (`bBusy=FALSE`).
tTimeout	TIME	Time until the internal ADS communication is aborted.

Outputs

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

Name	Type	Description
bBusy	BOOL	TRUE, as long as the function block is active.
bError	BOOL	TRUE if an error occurs during command execution.
nErrID	UDINT	Contains the ADS error code or the command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (system) v3.6.1

3.1.7 GETCURTASKINDEX

Outdated function block

This function block is outdated. Use the function GETCURTASKINDEXEX() [▶ 98] instead.

The function block GETCURTASKINDEX determines the task index of the task in which it is currently called.

To differentiate whether the current call occurs in the real-time context or from a cyclic PLC task, see the documentation for the function GETCURTASKINDEXEX [▶ 98]. For example, the automatic call of FB_init methods during initialization does not occur from a cyclic PLC task.

Inputs

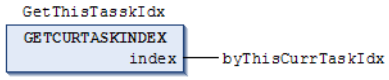
```
VAR_INPUT
(*none*)
END_VAR
```

Outputs

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

Name	Type	Description
index	BYTE	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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.1.8 FB_CreateGUID

The function block generates a new GUID. It is possible to get a list of different new GUIDs with one call if an array of type GUID is specified as buffer at the input.

Inputs

```
VAR_INPUT
    bExecute        : BOOL;
    sNetId          : T_AmsNetId;
    tTimeout        : TIME := DEFAULT_ADS_TIMEOUT;
    pGuidBuffer     : POINTER TO GUID;
    nGuidBufferSize : UDINT;
END_VAR
```

Name	Type	Description
bExecute	BOOL	The function block is activated by a rising edge at this input.
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.
pGuidBuffer	POINTER TO GUID	Indicates the address to the buffer for generated GUIDs. It is possible to specify the address on an ARRAY OF GUID.
nGuidBufferSize	UDINT	Indicates the size in bytes of the specified buffer.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrorId	UDINT	In the event of a set bError output returns the ADS error code [▶ 143].

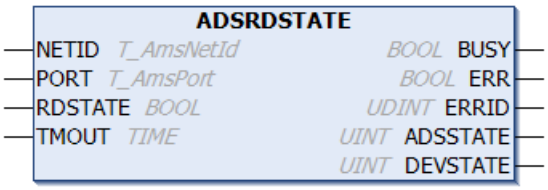
Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.4022	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.18.0

3.2 ADS function blocks

3.2.1 Control + State

3.2.1.1 ADSRDSTATE



The function block requests the state of an ADS device.

Inputs

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

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122]).
RDSTATE	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

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

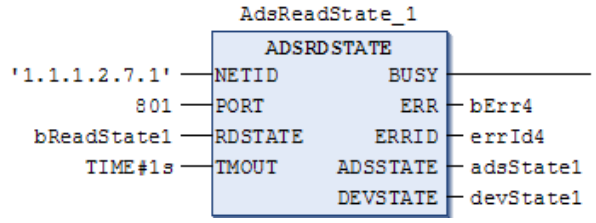
Name	Type	Description
BUSY	BOOL	This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the Timeout input. While BUSY = TRUE, no new command will be accepted at the inputs. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.
ADSSTATE	UDINT	Contains the state identification code of the ADS target device.
DEVSTATE	UDINT	Contains the specific state identification code of the ADS target device. The codes returned here are supplementary information specific to the ADS device.

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;
- ADSSTATE_SHUTDOWN = 12;
- ADSSTATE_SUSPEND = 13;
- ADSSTATE_RESUME = 14;
- ADSSTATE_CONFIG = 15;
- ADSSTATE_RECONFIG = 16;
- ADSSTATE_STOPPING = 17;
- ADSSTATE_INCOMPATIBLE = 18;
- ADSSTATE_EXCEPTION = 19;

Sample of calling the function block in FBD:

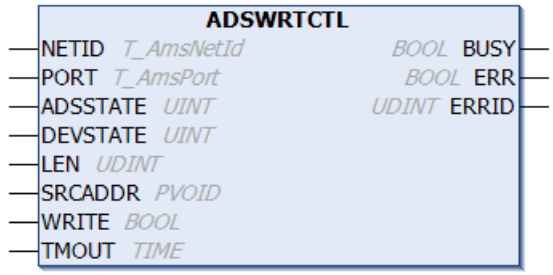


In this sample the PLC runtime system 1 (port no. 801) on the computer with network address 1.1.1.2.7.1 is asked about its state. The answer is adsState = 1 (IDLE) without supplementary code devState=0.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.1.2 ADSWRTCTL



This function 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.

Inputs

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

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122]).
ADSSTATE	UINT	State identification code of the ADS target device.
DEVSTATE	UINT	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	UDINT	Number of data to be written in bytes.
SRCADDR	PVOID	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	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

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;

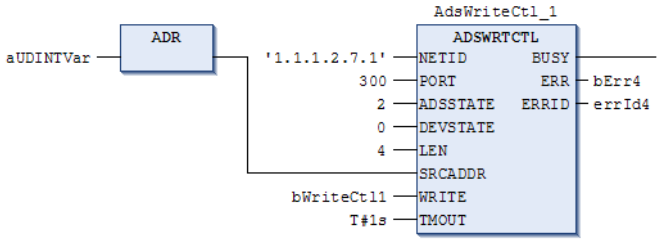
- ADSSTATE_START = 4;
- ADSSTATE_RUN= 5;
- ADSSTATE_STOP = 6;
- ADSSTATE_SAVECFG = 7;
- ADSSTATE_LOADCFG = 8;

Outputs

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

Name	Type	Description
BUSY	BOOL	This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the Timeout input. No new commands are accepted at the inputs as long as BUSY = TRUE. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.1.3 ADSRDDEVINFO

The function block reads the general device information.

Inputs

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

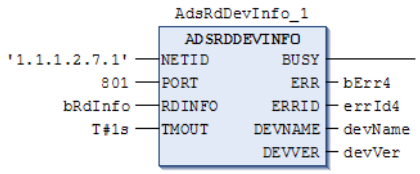
Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122]).
RDINFO	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

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

Name	Type	Description
BUSY	BOOL	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. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.
DEV NAME	STRING	Name of the ADS device
DEVVER	UDINT	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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.2 Indication + Response

3.2.2.1 Overview

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

- Request
- Indication
- Response
- Confirmation

The communication between an ADS device and a PLC task has the following sequence: 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) then carries out a corresponding service. The service to be carried out is encrypted via the index-group/offset parameter. Next the PLC task sends a response to the ADS device. The response is registered as confirmation by the ADS source device.

The ADS devices are addressed via a port address and a network address (NETID). (Port address of the PLC task = _TaskInfo.AdsPort)

In order for a request to be routed to the PLC task, the most significant bit (e.g. 0x80000001) must be entered in the index group parameter when the request is made.

Communication via IndexGroup 0x80000000 - 0x80FFFFFF

Only one instance of the indication and response function block can 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.

Service	Name	Description
READ	ADSREADIND [▶ 27]	ADSREAD Indication
	ADSREADRES [▶ 36]	ADSREAD Response
WRITE	ADSWRITEIND [▶ 30]	ADSWRITE Indication
	ADSWRITERES [▶ 37]	ADSWRITE Response
READ & WRITE	ADSRDWRTIND [▶ 33]	ADS-READ & WRITE Indication
	ADSRDWRTRES [▶ 38]	ADS-READ & WRITE Response

FiFos

Each PLC task has 3 FIFOS in which the incoming requests (indications) are first stored. This means that there is an ADSREADIND FIFO, an ADSWRITEIND FIFO and an ADSRDWRTIND FIFO.
In each FIFO a maximum of 10 indications can be stored, until these were processed (until 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 analyzed 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.

Communication via IndexGroup 0x8n000000 - 0x8nFFFFFF

To realize more than one client-server communication per PLC task, the following Indication function blocks are required. These are extended by the possibility to specify a desired range of the IndexGroup.

This way, requests are filtered and only desired areas are responded to.

There are 16 freely selectable ranges available:
0x80000000 - 0x80FFFFFF
0x81000000 - 0x81FFFFFF

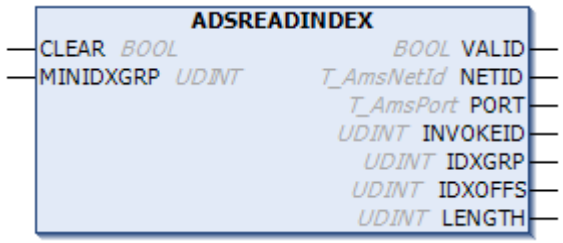
…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

To specify such a range of the index group at an Indication function block, the index group value with which the selected range begins is specified at input MINIDXGRP.
Example: With MINIDXGRP:=16#85000000 all requests are filtered and requests with an index group in the range 0x85000000 - 0x85FFFFFF are registered as Indication.

Service	Name	Description
READ	ADSREADINDEX [▶ 28]	ADSREAD Indication with indication of the index group
	ADSREADRES [▶ 36]	ADSREAD Response
WRITE	ADSWRITEINDEX [▶ 31]	ADSWRITE Indication with indication of the index group
	ADSWRITERES [▶ 37]	ADSWRITE Response
READ & WRITE	ADSRDWRTINDEX [▶ 34]	ADS-READ & WRITE Indication with indication of the index group
	ADSRDWRTRES [▶ 38]	ADS-READ & WRITE Response

3.2.2.2 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 reported as processed via a positive edge at the CLEAR input. A negative 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 [▶ 36] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter is used by the source device to assign the responses to the requests and is also sent back to the source device as a parameter.

Inputs

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

Name	Type	Description
CLEAR	BOOL	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.

Outputs

```
VAR_OUTPUT
    VALID    : BOOL;
    NETID    : T_AmsNetId;
    PORT     : T_AmsPort;
```

```
    INVOKEID : UDINT;
    IDXGRP   : UDINT;
    IDXOFFS  : UDINT;
    LENGTH   : UDINT;
END_VAR
```

Name	Type	Description
VALID	BOOL	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 positive edge at the CLEAR input.
NETID	T_AmsNetId	String containing the AMS network ID of the source device, from which the ADS command was sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Contains the port number of the ADS source device, from which the ADS command was sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeID is specified from the source device and serves to identify the commands.
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service.
IDXOFFS	UDINT	Index offset number (32-bit, unsigned) of the requested ADS service.
LENGTH	UDINT	Number of data to be read in bytes.

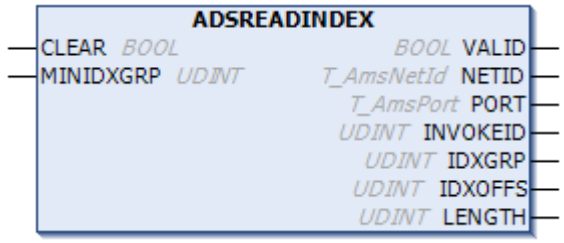
Example of calling the block in ST:

- Example with AdsReadInd /AdsReadRes function blocks [▶ 132]

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.2.3 ADSREADINDEX



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 reported as processed via a positive edge at the CLEAR input. A negative 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 [▶ 36] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter is used by the source device to assign the responses to the requests and is also sent back to the source device as a parameter.

Compared to the function block ADSREADIND [▶ 27] there is the possibility to specify a desired range of the IndexGroup via an additional input.

This way, requests are filtered and only desired areas are responded to.

There are 16 freely selectable ranges available:
0x80000000 - 0x80FFFFFF
0x81000000 - 0x81FFFFFF

…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

To specify such a range of the index group at an Indication function block, the index group value with which the selected range begins is specified at input MINIDXGRP.
Example: With MINIDXGRP:=16#85000000 all requests are filtered and requests with an index group in the range 0x85000000 - 0x85FFFFFF are registered as Indication.

Inputs

```
VAR_INPUT
    CLEAR      : BOOL;
    MINIDXGRP  : UDINT;
END_VAR
```

Name	Type	Description
CLEAR	BOOL	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.
MINIDXGRP	UDINT	This input allows filtering the requests by IndexGroup ranges. Specification of the IndexGroup value with which the selected range begins.

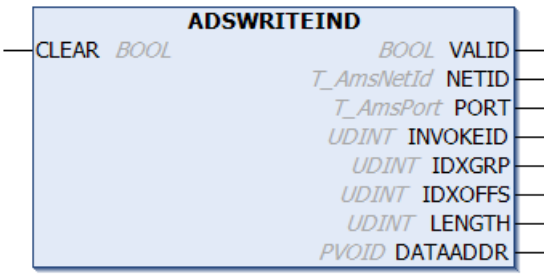
Outputs

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

Name	Type	Description
VALID	BOOL	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 positive edge at the CLEAR input.
NETID	T_AmsNetId	String containing the AMS network ID of the source device, from which the ADS command was sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Contains the port number of the ADS source device, from which the ADS command was sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeID is specified from the source device and serves to identify the commands.
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service.
IDXOFFS	UDINT	Index offset number (32-bit, unsigned) of the requested ADS service.
LENGTH	UDINT	Number of data to be read in bytes.

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.35	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.26.0

3.2.2.4 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 reported as processed via a positive edge at the CLEAR input. 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 [▶ 37] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter is used by the source device to assign the responses to the requests and is also sent back to the source device as a parameter.

Inputs

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

Name	Typ	Beschreibung
CLEAR	BOOL	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.

Outputs

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

Name	Type	Description
VALID	BOOL	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 positive edge at the CLEAR input.
NETID	T_AmsNetId	String containing the AMS network ID of the source device, from which the ADS command was sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Contains the port number of the ADS source device, from which the ADS command was sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeID is specified from the source device and serves to identify the commands.
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service.
IDXOFFS	UDINT	Index offset number (32-bit, unsigned) of the requested ADS service.
LENGTH	UDINT	Length of the written data in bytes.
DATAADDR	PVOID	Address of the data buffer in which the written data is located.

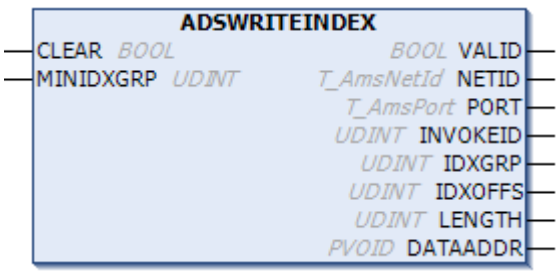
Example of calling the block in ST:

- Example with AdsWriteInd/AdsWriteRes function blocks [▶ 134]

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.2.5 ADSWRITEINDEX



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 reported as processed via a positive edge at the CLEAR input. 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 [▶ 37] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter is used by the source device to assign the responses to the requests and is also sent back to the source device as a parameter.

Compared to the ADSWRITEIND [▶ 30] function block there is the possibility to specify a desired range of the IndexGroup via an additional input.

This way, requests are filtered and only desired areas are responded to.

There are 16 freely selectable ranges available:
0x80000000 - 0x80FFFFFF
0x81000000 - 0x81FFFFFF

…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

To specify such a range of the index group at an Indication function block, the index group value with which the selected range begins is specified at input MINIDXGRP.
Example: With MINIDXGRP:=16#85000000 all requests are filtered and requests with an index group in the range 0x85000000 - 0x85FFFFFF are registered as Indication.

Inputs

```
VAR_INPUT
    CLEAR      : BOOL;
    MINIDXGRP  : UDINT;
END_VAR
```

Name	Type	Description
CLEAR	BOOL	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.
MINIDXGRP	UDINT	This input allows filtering the requests by IndexGroup ranges. Specification of the IndexGroup value with which the selected range begins.

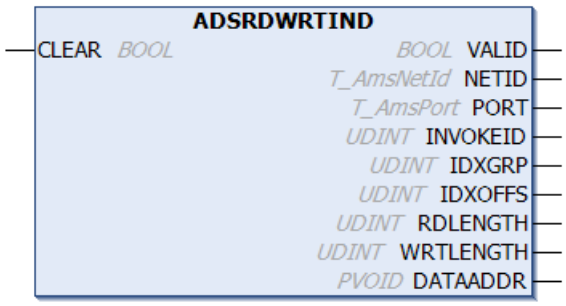
Outputs

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

Name	Type	Description
VALID	BOOL	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 positive edge at the CLEAR input.
NETID	T_AmsNetId	String containing the AMS network ID of the source device, from which the ADS command was sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Contains the port number of the ADS source device, from which the ADS command was sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeID is specified from the source device and serves to identify the commands.
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service.
IDXOFFS	UDINT	Index offset number (32-bit, unsigned) of the requested ADS service.
LENGTH	UDINT	Length of the written data in bytes.
DATAADDR	PVOID	Address of the data buffer in which the written data is located.

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.35	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.26.0

3.2.2.6 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 reported as processed via a positive edge at the CLEAR input. 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 [▶ 38] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter is used by the source device to assign the responses to the requests and is also sent back to the source device as a parameter.

Inputs

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

Name	Type	Description
CLEAR	BOOL	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.

Outputs

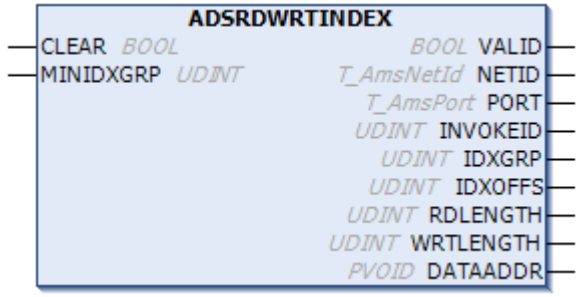
```
VAR_OUTPUT
    VALID     : BOOL;
    NETID     : T_AmsNetId;
    PORT      : T_AmsPort;
    INVOKEID  : UDINT;
    IDXGRP    : UDINT;
    IDXOFFS   : UDINT;
    RDLENGTH  : UDINT;
    WRTLENGTH : UDINT;
    DATAADDR  : PVOID;
END_VAR
```

Name	Type	Description
VALID	BOOL	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 positive edge at the CLEAR input.
NETID	T_AmsNetId	String containing the AMS network ID of the source device, from which the ADS command was sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Contains the port number of the ADS source device, from which the ADS command was sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service.
IDXOFFS	UDINT	Index offset number (32-bit, unsigned) of the requested ADS service.
RDLENGTH	UDINT	Length of the data to be read in bytes.
WRTLENGTH	UDINT	Length of the written data in bytes.
DATAADDR	PVOID	Address of the data buffer in which the written data is located.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.2.7 ADSRDWRTINDEX



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 reported as processed via a positive edge at the CLEAR input. 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 [▶ 38] function block. The PORT and NETID parameters can be used to address the source device for this purpose. The INVOKEID parameter is used by the source device to assign the responses to the requests and is also sent back to the source device as a parameter.

Compared to the ADSRDWRTIND [▶ 33] function block there is the possibility to specify a desired range of the IndexGroup via an additional input.

This way, requests are filtered and only desired areas are responded to.

There are 16 freely selectable ranges available:
0x80000000 - 0x80FFFFFF
0x81000000 - 0x81FFFFFF

…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

To specify such a range of the index group at an Indication function block, the index group value with which the selected range begins is specified at input MINIDXGRP.
Example: With MINIDXGRP:=16#85000000 all requests are filtered and requests with an index group in the range 0x85000000 - 0x85FFFFFF are registered as Indication.

Inputs

```
VAR_INPUT
    CLEAR      : BOOL;
    MINIDXGRP  : UDINT;
END_VAR
```

Name	Type	Description
CLEAR	BOOL	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.
MINIDXGRP	UDINT	This input allows filtering the requests by IndexGroup ranges. Specification of the IndexGroup value with which the selected range starts.

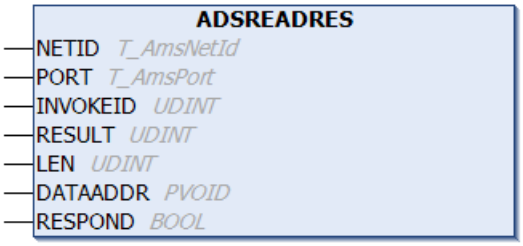
Outputs

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

Name	Type	Description
VALID	BOOL	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 positive edge at the CLEAR input.
NETID	T_AmsNetId	String containing the AMS network ID of the source device, from which the ADS command was sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Contains the port number of the ADS source device, from which the ADS command was sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service.
IDXOFFS	UDINT	Index offset number (32-bit, unsigned) of the requested ADS service.
RDLENGTH	UDINT	Length of the data to be read in bytes.
WRTLENGTH	UDINT	Length of the written data in bytes.
DATAADDR	PVOID	Address of the data buffer in which the written data is located.

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.35	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.26.0

3.2.2.8 ADSREADRES



The ADSREADRES function block acknowledges Indications of a PLC task. A response is sent to the ADS source device via a positive edge on the RESPOND input. The source device is addressed via the PORT and NETID parameters. The INVOKEID parameter is used by the source device to assign the responses to the requests 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.

Inputs

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

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the source device, to which the ADS command is to be sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS source device to which the response should be sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeID is specified from the source device and serves to identify the commands.
RESULT	UDINT	ADS error code [▶ 143] or command-specific error code to be sent to the source device.
LEN	UDINT	Number of data to be read in bytes.
DATAADDR	PVOID	Address of the data buffer, which should be read.
RESPOND	BOOL	The function block is activated by a positive edge at this input.

Outputs

```
VAR_OUTPUT
(*none*)
END_VAR
```

Example of calling the block in ST:

- Example with AdsReadInd /AdsReadRes function blocks [▶ 132]

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.2.9 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.

Inputs

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

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the source device, to which the ADS command is to be sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS source device to which the ADS command is to be sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeId is specified from the source device and serves to identify the commands.
RESULT	UDINT	ADS error code [▶ 143] or command-specific error code to be sent to the source device.
RESPOND	BOOL	The function block is activated by a positive edge at this input.

Outputs

```
VAR_OUTPUT
(*none*)
END_VAR
```

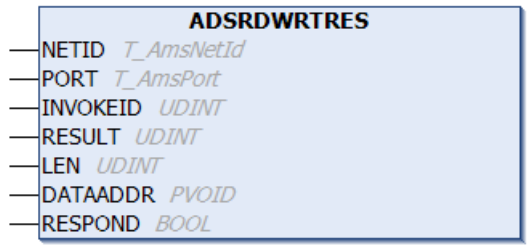
Example of calling the block in ST:

- Example with AdsWriteInd/AdsWriteRes function blocks [▶ 134]

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.2.10 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 [▶ 33] function block. An error code can be returned to the ADS source device via the RESULT parameter.

Inputs

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

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the source device, to which the ADS command is to be sent (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS source device to which the ADS command is to be sent (type: T_AmsPort [▶ 122]).
INVOKEID	UDINT	Handle of the command, which was sent. The InvokeID is specified from the source device and serves to identify the commands.
RESULT	UDINT	ADS error code [▶ 143] or command-specific error code to be sent to the source device.
LEN	UDINT	Length, in bytes, of the read data.
DATAADDR	PVOID	Address of the data buffer, in which the read data is located.
RESPOND	BOOL	The function block is activated by a positive edge at this input.

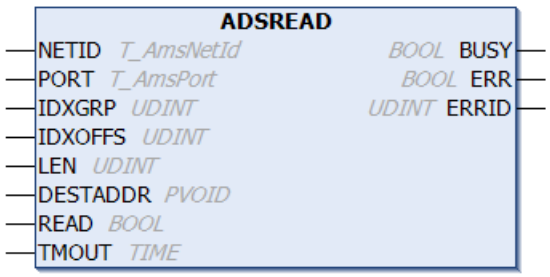
Outputs

```
VAR_OUTPUT
(*none*)
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.3 ADSREAD



The function block executes an ADS read command to request data from an ADS device.

Outdated response data

After a disconnection, the old response data is output when the connection is reconnected. To prevent this, be careful not to use the same ADS-Read instance for multiple targets.

Inputs

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

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122])
IDXGRP	UDINT	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	UDINT	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	UDINT	Number of data to be read in bytes
DESTADDR	PVOID	Address of the buffer that is to receive the read data. The programmer is responsible for dimensioning the buffer such that it can accommodate LEN bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
READ	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

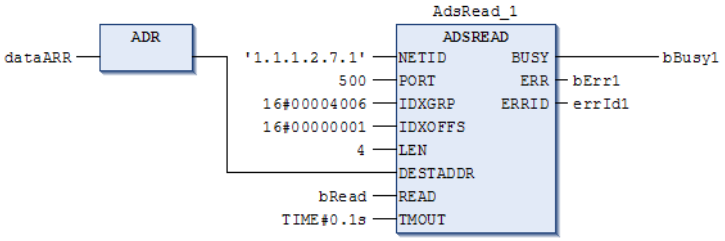
```
VAR_OUTPUT
    BUSY     : BOOL;
    ERR      : BOOL;
    ERRID    : UDINT;
END_VAR
```

Name	Type	Description
BUSY	BOOL	This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the Timeout input. No new commands are accepted at the inputs as long as BUSY = TRUE. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in ST:

- Example with AdsRead function block [▶ 135]

Example of calling the block in FBD:

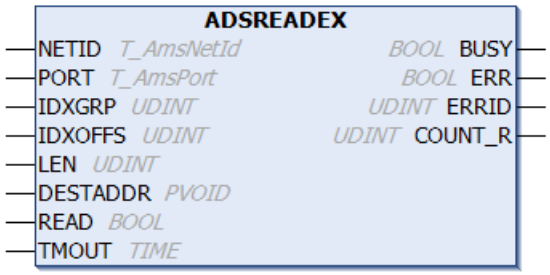


Here the error status of axis no. 6, as an element with a size of 4 bytes, is interrogated and written into the dataArr buffer. The IDXGRP 00004006 (hex) and the IDXOFFS 00000001 (hex) can be found in the NC-ADS documentation.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.4 ADSREADEX



The function block executes an ADS read command to request data from an ADS device. The function block has the same functionality as the ADSREAD function block; in addition it returns the number of actually read data bytes as parameter.

Inputs

```
VAR_INPUT
    NETID    : T_AmsNetId;
    PORT     : T_AmsPort;
    IDXGRP   : UDINT;
    IDXOFFS  : UDINT;
    LEN      : UDINT;
    DESTADDR : PVOID;
```

```
    READ     : BOOL;
    TMOUT    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122])
IDXGRP	UDINT	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	UDINT	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	UDINT	Number of data to be read in bytes
DESTADDR	PVOID	Address of the buffer that is to receive the read data. The programmer is responsible for dimensioning the buffer such that it can accommodate LEN bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
READ	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

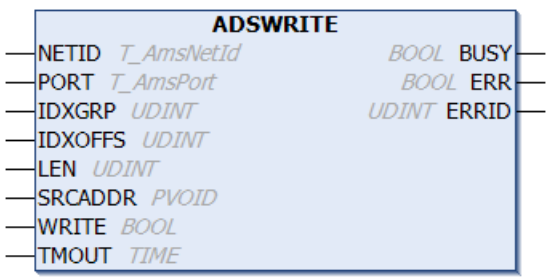
```
VAR_OUTPUT
    BUSY     : BOOL;
    ERR      : BOOL;
    ERRID    : UDINT;
    COUNT_R  : UDINT;
END_VAR
```

Name	Type	Description
BUSY	BOOL	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. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.
COUNT_R	UDINT	Number of successfully read data bytes

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.5 ADSWRITE



Block for executing an ADS write command for transferring data to an ADS device.

Inputs

```
VAR_INPUT
    NETID   : T_AmsNetId;
    PORT    : T_AmsPort;
    IDXGRP  : UDINT;
    IDXOFFS : UDINT;
    LEN     : UDINT;
    SRCADDR : PVOID;
    WRITE   : BOOL;
    TMOUT   : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122])
IDXGRP	UDINT	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	UDINT	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	UDINT	Number of data to be read in bytes
SRCADDR	PVOID	Address of the buffer from which the data to be written is to be fetched. The programmer is responsible for dimensioning the buffer such 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	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

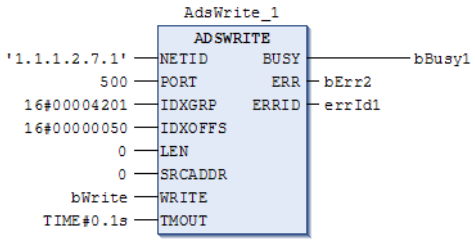
```
VAR_OUTPUT
    BUSY    : BOOL;
    ERR     : BOOL;
    ERRID   : UDINT;
END_VAR
```

Name	Type	Description
BUSY	BOOL	This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the Timeout input. No new commands are accepted at the inputs as long as BUSY = TRUE. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in ST:

- Example with AdsWrite function block [▶ 136]

Example of calling the block in FBD

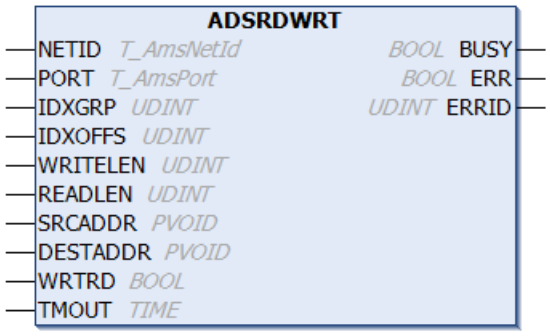


NC axis no. 1 is here deactivated through a write instruction with IDXGRP 00004201 (hex) and the IDXOFFS 00000050 (hex). To activate the axis another write instruction with the IDXOFFS 00000051 (hex) must be given. Since this write instruction does not require any parameters, the inputs LEN and SRCADDR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

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.6 ADSRDWRT



The function block executes a combined ADS read-write command. Data is transmitted to an ADS device (write) and its response data read with one call.

Inputs

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

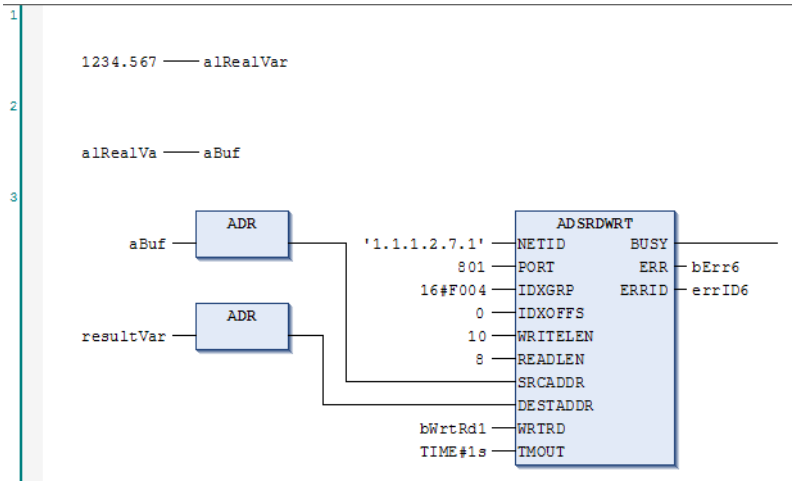
Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122])
IDXGRP	UDINT	Index group number (32-bit, unsigned) of the requested ADS service. This value is to be found in the ADS table of the addressed device.
IDXOFFS	UDINT	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	UDINT	Number of data to be written in bytes
READLEN	UDINT	Number of data to be read in bytes
SRCADDR	PVOID	Address of the buffer from which the data to be written is to be fetched. The programmer is responsible for dimensioning the buffer such 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	PVOID	Address of the buffer that is to receive the read data. The programmer is responsible for dimensioning the buffer such that it can accommodate READLEN bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
WRTRD	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

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

Name	Type	Description
BUSY	BOOL	This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the Timeout input. No new commands are accepted at the inputs as long as BUSY = TRUE. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the block in FBD:

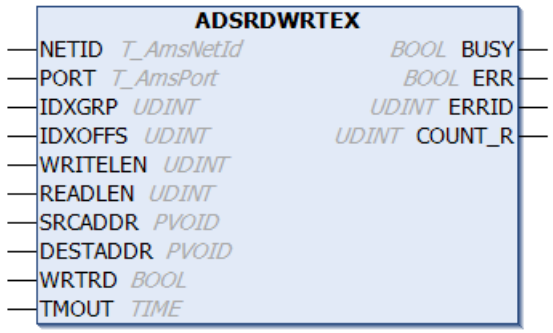


The value of the variable with the name “aLRealVar” is here read from the PLC which is running on the computer with the Net-Id 1.1.1.2.7.1. For this purpose, the computer address mentioned, the port number of the PLC’s first run-time system, the index group, and the index offset for reading the variable 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.2.7 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 supplies the number of actually read data bytes as parameter.

Inputs

```
VAR_INPUT
    NETID    : T_AmsNetId;
    PORT     : T_AmsPort;
    IDXGRP   : UDINT;
    IDXOFFS  : UDINT;
    WRITELEN : UDINT;
    READLEN  : UDINT;
    SRCADDR  : PVOID;
    DESTADDR : PVOID;
```

```
    WRTRD    : BOOL;
    TMOUT    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device (type: T_AmsPort [▶ 122])
IDXGRP	UDINT	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	UDINT	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	UDINT	Number of data to be written in bytes
READLEN	UDINT	Number of data to be read in bytes
SRCADDR	PVOID	Address of the buffer from which the data to be written is to be fetched. The programmer is responsible for dimensioning the buffer such 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	PVOID	Address of the buffer that is to receive the read data. The programmer is responsible for dimensioning the buffer such that it can accommodate READLEN bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
WRTRD	BOOL	The ADS command is triggered by a rising edge at this input.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

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

Name	Type	Description
BUSY	BOOL	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. Note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ERRID. If the function block has a timeout error, ERR is TRUE and ERRID is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ERRID	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.
COUNT_R	UDINT	Number of successfully read data bytes

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3 File function blocks

3.3.1 FB_EOF

The function block can check whether the end of the file was reached.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

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

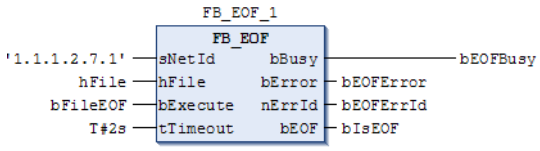
Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
bEOF	BOOL	This output is set when the end of the file is reached.

Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the function 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.2 FB_FileOpen

The function block creates a new file or opens an existing file for editing.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
sPathName	T_MaxString	Storage path and file name of the file to be opened. The path can only point to the local file system of the computer. Network paths cannot be specified here (type: T_MaxString [▶ 124]).
nMode	DWORD	Mode for opening the file.
ePath	E_OpenPath	This input can be used to select a TwinCAT system path on the target device for opening the file (type: E_OpenPath [▶ 118]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Predefined opening modes for nMode

Mode for opening the file. 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

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
hFile	UINT	Contains the file handle created for the file when opening has been successful. This handle is then transferred to the other file function blocks as ID for the file to be edited.

Error codes for hFile

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

Information:

In the opening mode, a maximum of 3 parameters may be ORed:

Mode = [Parameter1] OR [Parameter2] OR [Paramerter3]
Parameter1 may have only a 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
FOPEN_MODEAPPEND OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEAPPEND OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY

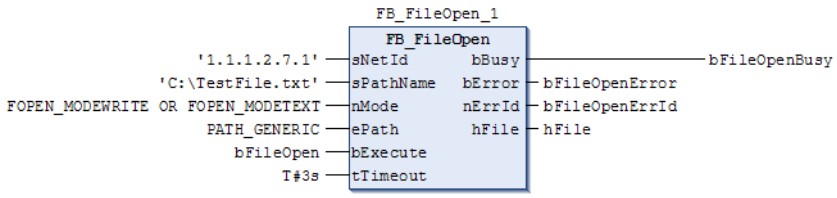
All other combinations are wrong. Examples of invalid opening modes:
FOPEN_MODEBINARY OR FOPEN_MODETEXT
FOPEN_MODEWRITE OR FOPEN_MODEAPPEND

Example of calling the block in ST:

- File access from the PLC [▶ 138]

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.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.3 FB_FileClose

The function block closes the file, thereby putting it in a defined state for further processing by other programs.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	Handle of the file to be closed.
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command specific error code for nErrId

Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

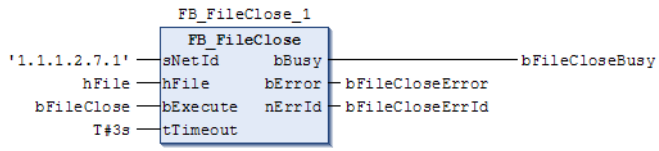
Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the block in ST:

- File access from the PLC [▶ 138]

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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.4 FB_FileLoad

The contents of a file can be read out with this function block. The file is opened implicitly in binary mode, the contents are read out and the file is then closed again.

Inputs

```
VAR_INPUT
    sNetId     : T_AmsNetId;
    sPathName  : T_MaxString;
    pReadBuff  : PVOID;
    cbReadLen  : UDINT;
    bExecute   : BOOL;
    tTimeout   : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
sPathName	T_MaxString	Storage path and file name of the file to be opened. The path can only point to the local file system of the computer. Network paths cannot be specified here (type: T_MaxString [▶ 124])
pReadBuff	PVOID	Address of the buffer into which the data are to be read. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
cbReadLen	UDINT	Number of bytes to be read.
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the internal ADS command.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
cbRead	UDINT	Number of currently read bytes

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.4022.0	PC or CX (x86, x64, ARM)	Tc2_System (System) >= v3.4.22.0

3.3.5 FB_FileGets

The function block reads strings from a file. The string is read up to and including the line feed character, or up to the end of the file or the maximum permitted length of sLine. The zero termination is appended automatically. The file must have been opened in text mode.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

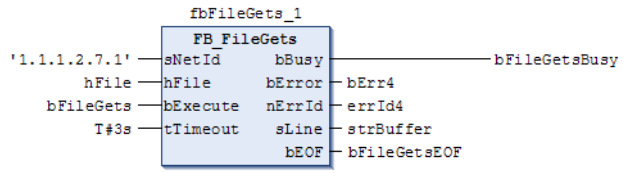
```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    sLine   : T_MaxString;
    bEOF    : BOOL;
END_VAR
```

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
sLine	T_MaxString	String that was read (type: T_MaxString [▶ 124]).
bEOF	BOOL	This output is set if the end of the file was reached and no further data bytes could be read (cbRead=0). This output is not set if further data bytes could be read (cbRead>0).

Command-specific error code	Possible cause
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 function block in FBD:

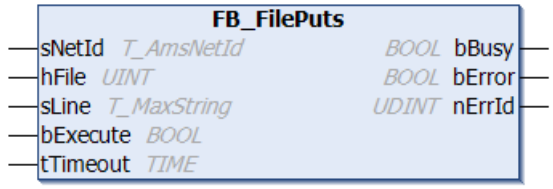
```
PROGRAM Test
VAR
    fbFileGets        : FB_FileGets;
    hFile             : UINT;
    bFileGets         : BOOL;
    bFileGetsBusy     : BOOL;
    bFileGetsError    : BOOL;
    nFileGetsErrorId  : UDINT;
    strBuffer         : STRING;
    bFileGetsEOF      : BOOL;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.6 FB_FilePuts



The function block writes strings into a file. The string is written to the file up to the null termination, but without the null character. The file must have been opened in text mode.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
sLine	T_MaxString	String to be written into the file (type: T_MaxString [▶ 124]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

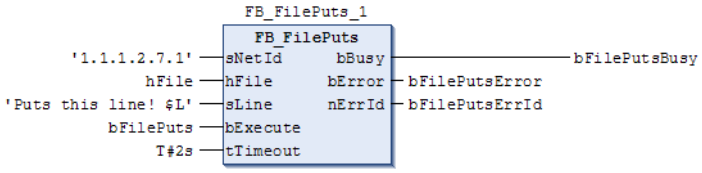
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the function block in FBD:

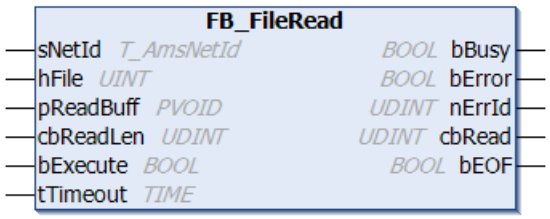
```
PROGRAM Test
VAR
    fbFilePuts       : FB_FilePuts;
    hFile            : UINT;
    bFilePuts        : BOOL;
    bFilePutsBusy    : BOOL;
    bFilePutsError   : BOOL;
    nFilePutsErrorId : UDINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.7 FB_FileRead



With this function block the contents of an already opened file can be read. Before a read access, the file must have been opened in the corresponding mode. In addition to the `FOPEN_MODEREAD`, the appropriate format (`FOPEN_MODEBINARY` or `FOPEN_MODETEXT`) is also important to achieve the desired result.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
pReadBuff	PVOID	Address of the buffer into which the data are to be read. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
cbReadLen	UDINT	Number of bytes to be read
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
cbRead	UDINT	Number of currently read bytes
bEOF	BOOL	This output is set if the end of the file was reached and no further data bytes could be read (cbRead=0). This output is not set if further data bytes could be read (cbRead>0).

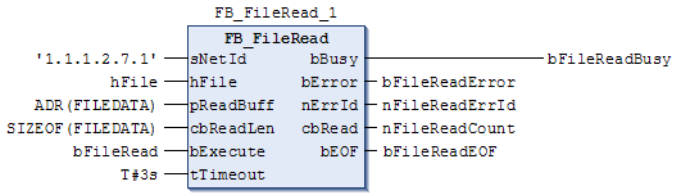
Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70A	No memory for read buffer.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the block in ST:

- File access from the PLC [▶ 138]

Example of calling the function 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
```

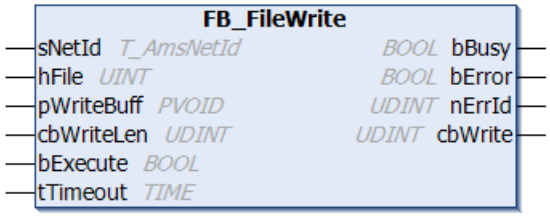


After a rising edge at bExecute and successful execution of the read instruction the currently read bytes from the file are found in FILEDATA. The parameter cbRead can be used to determine how many bytes were actually read during the last read operation.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.8 FB_FileWrite



The function block writes data into a file. For write access the file must have been opened in the corresponding mode, and it must be closed again for further processing by external programs.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
pWriteBuff	PVOID	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	UDINT	Number of bytes to be written
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

```
VAR_OUTPUT
    bBusy    : BOOL;
    bError   : BOOL;
```

```
    nErrId   : UDINT;
    cbWrite  : UDINT;
END_VAR
```

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
cbWrite	UDINT	Contains the number of the last successfully written data bytes. A write error indicates that the number of successfully written data bytes is less than the requested length (cbWriteLen) or zero. A write error can occur if the data carrier is full, for example. If a write error occurs, the bError and nErrID outputs are not set. Since the PLC application knows the number of data bytes to be written, it can compare the actual written length with the requested length and detect write errors. When a write error occurs, the internal file pointer has an undefined position.

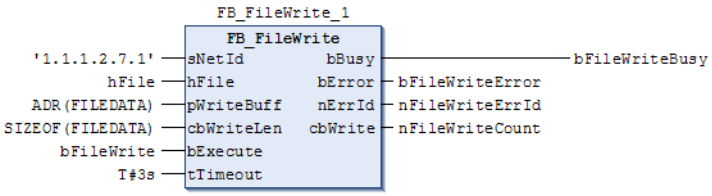
Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the block in ST:

- File access from the PLC [▶ 138]

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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.9 FB_FileSeek

The function block sets the file pointer of an open file to a definable position.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
nSeekPos	DINT	New position of the file pointer
eOrigin	E_SeekOrigin	Relative position, to which the file pointer is to be moved (type: E_SeekOrigin [▶ 118]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

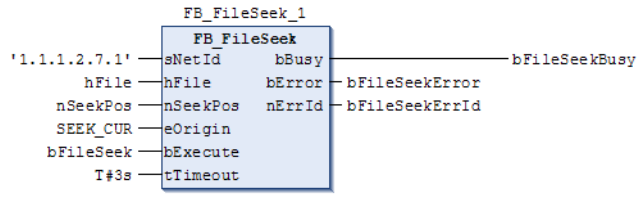
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the function 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.10 FB_FileTell

The function block determines the current position of the file pointer. The position indicates the relative offset from the start of the file.

Note that in files opened in "Append at end of file" mode, the current position is determined by the last I/O operation, not by the position of the next write access. After a read operation, for example, the file pointer is at the position where the next read access will take place, not at the position where the next write access will take place. In append mode, the file pointer is always moved to the end before the write operation.

If no previous I/O operation was performed and the file was opened in append mode, the file pointer is at the start of the file.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
hFile	UINT	File handle, which was generated when the function block FB_FileOpen was created.
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

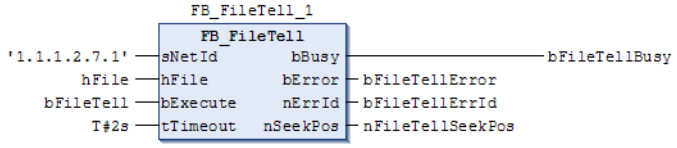
```
VAR_OUTPUT
    bBusy    : BOOL;
    bError   : BOOL;
    nErrId   : UDINT;
    nSeekPos : DINT;(* On error, nSEEKPOS returns -1 *)
END_VAR
```

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.
nSeekPos	DINT	Returns the current position of the file pointer.

Command-specific error code	Possible cause
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Example of calling the function block in FBD:

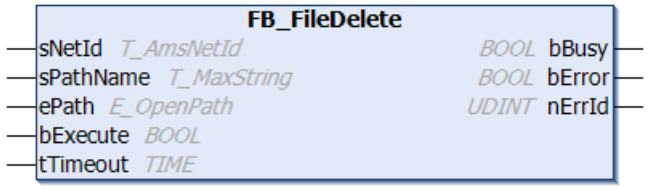
```
PROGRAM Test
VAR
    fbFileTell         : FB_FileTell;
    hFile              : UINT;
    bFileTell          : BOOL;
    bFileTellBusy      : BOOL;
    bFileTellError     : BOOL;
    nFileTellErrorId   : UDINT;
    nFileTellSeekPos   : DINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.11 FB_FileDelete



The function block deletes a file from the data storage device.

Inputs

```
VAR_INPUT
    sNetId    : T_AmsNetId;
    sPathName : T_MaxString;(* file path and name *)
    ePath     : E_OpenPath := PATH_GENERIC;
```

```
    bExecute  : BOOL;
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
sPathName	T_MaxString	File name, including the full path (type: T_MaxString [▶ 124]).
ePath	E_OpenPath	This input can be used to select a TwinCAT system path on the target device for opening the file (type: E_OpenPath [▶ 118]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

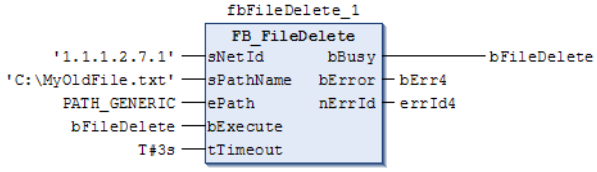
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command-specific error code	Possible cause
0x70C	File not found. Invalid sPathName or ePath parameter.

Example of calling the function 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.12 FB_FileRename

This function block can be used to rename a file.

Inputs

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

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
sOldName	T_MaxString	Old file name (type: T_MaxString [▶ 124])
sNewName	T_MaxString	New file name (type: T_MaxString [▶ 124])
ePath	E_OpenPath	This input can be used to select a TwinCAT system path on the target device for opening the file (type: E_OpenPath [▶ 118]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

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

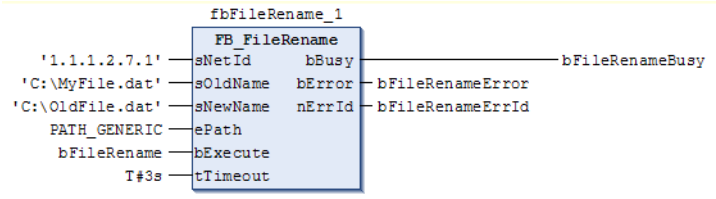
Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command-specific error code	Possible cause
0x70C	File not found. Invalid sOldName, sNewName or ePath parameter.

Example of calling the function 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.13 FB_CreateDir

This function block can be used to create new directories on the data storage device.

Inputs

```
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
```

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
sPathName	T_MaxString	Name of the new directory. When the function block is called, the only option is to create a new directory (type: T_MaxString [▶ 124]).
ePath	E_OpenPath	This input can be used to select a TwinCAT system path for the new directory on the target device (type: E_OpenPath [▶ 118]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command-specific error code	Possible cause
0x723	Folder is already 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.

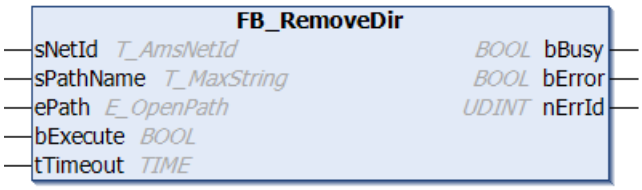
At bBootFolder = TRUE a directory in the *..\TwinCAT\Boot* directory can be created or deleted.

```
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 type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.3.14 FB_RemoveDir



This function block can be used to delete a directory from the data storage device. A directory containing files cannot be deleted.

Inputs

```
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
```

Name	Type	Description
sNetId	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
sPathName	T_MaxString	Directory name to be deleted. When the function block is called, only one directory can be deleted. The last component of sPathName must contain the directory name to be deleted (type: T_MaxString [▶ 124]).
ePath	E_OpenPath	This input can be used to select a TwinCAT system path for deleting the directory on the target device (type: E_OpenPath [▶ 118]).
bExecute	BOOL	The function block is activated by a rising edge at this input.
tTimeout	TIME	States the length of the timeout that may not be exceeded by execution of the ADS command.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be created.
bError	BOOL	If an error occurs during command execution, this output is set, once the bBusy output has been reset.
nErrId	UDINT	Returns the ADS error code [▶ 143] or the command-specific error code when the bError output is set.

Command-specific error code	Possible cause
0x70C	Folder not found or invalid sPathName or ePath parameter.

Example in ST:

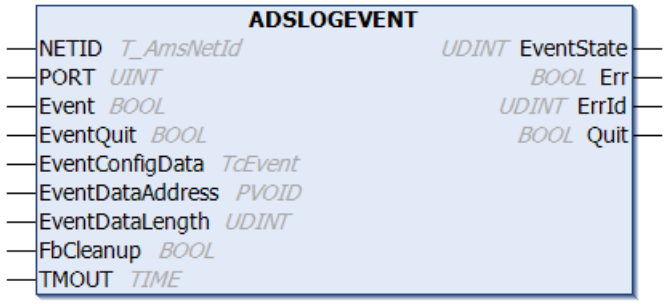
See description of FB_CreateDir [▶ 65].

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.4 EventLogger function blocks

3.4.1 ADSLOGEVENT



This function block allows the sending and acknowledgment of messages to the TwinCAT EventLogger.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

The TwinCAT EventLogger was replaced by the successor TwinCAT 3 EventLogger. The older TwinCAT EventLogger is supported by TwinCAT 3 up to version 3.1.4024. Newer TwinCAT versions (>= 3.1.4026.0) only support the newer TwinCAT 3 EventLogger. PLC function blocks for this can be found in the PLC library Tc3_EventLogger.

Inputs

```
VAR_INPUT
    NETID             : T_AmsNetId;
    PORT              : T_AmsPort;
    Event             : BOOL;
    EventQuit         : BOOL;
    EventConfigData   : TcEvent;
    EventDataAddress  : PVOID;
    EventDataLength   : UDINT;
    FbCleanup         : BOOL;
    TMOUT             : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

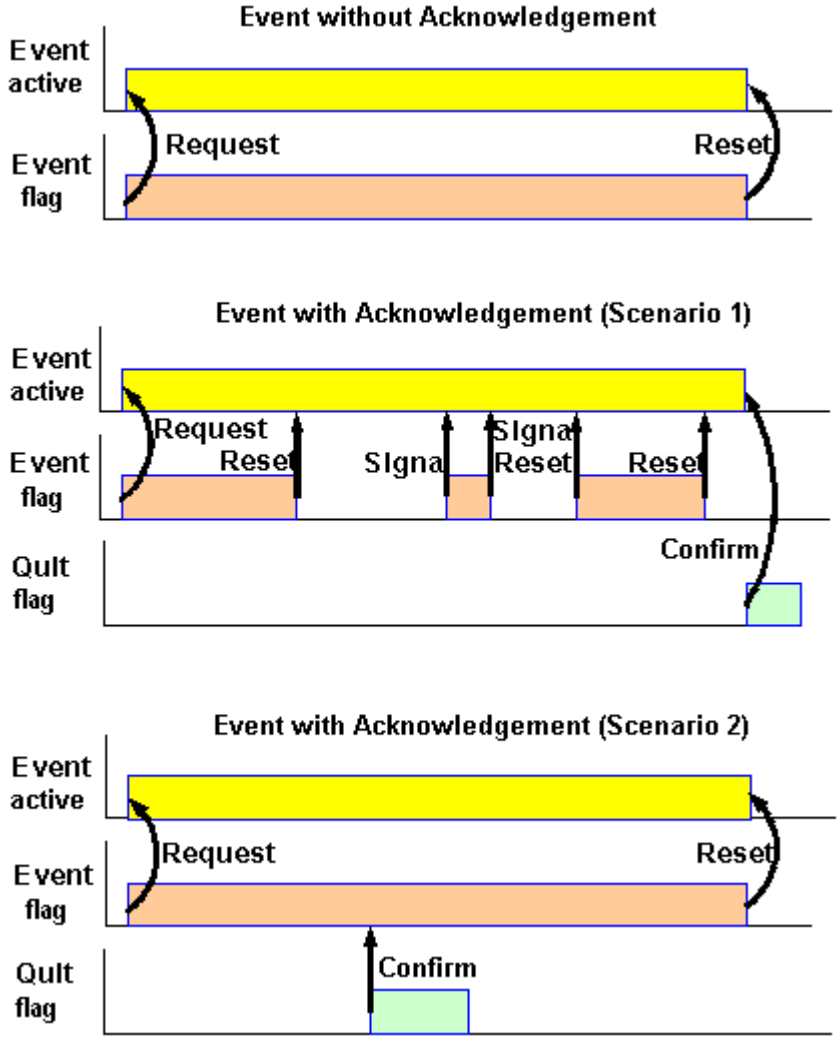
Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
PORT	T_AmsPort	Port number of the ADS device. The TwinCAT EventLogger has the port number 110 (type: T_AmsPort [▶ 122]).
Event	BOOL	The "coming" of the event is signaled with the rising edge, the "going" of the event with the falling edge.
EventQuit	BOOL	The event is acknowledged with the rising edge.
EventConfig Data	TcEvent	Data structure with the event parameters (type: TcEvent [▶ 124]).
EventData Address	PVOID	Address with the data to be sent with the event.
EventData Length	UDINT	Length of the data to be sent with the event.
FbCleanup	BOOL	If TRUE, the function block is completely initialized.
TMOUT	TIME	Indicates the time before the function is canceled.

Outputs

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

Name	Type	Description
EventState	UDINT	State of the event.
Err	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ErrId. If the function block has a timeout error, Err is TRUE and ErrId is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
ErrId	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.
Quit	BOOL	Acknowledges the event.

Acknowledge messages



The upper figure represents the general sequence.

In the case of messages not requiring acknowledgment, the event is announced with the rising edge at the event input of the function block and is thus active in the EventLogger. The falling edge at the event input initiates the reset. This signal deletes the event in the EventLogger again.

In the case of messages requiring acknowledgment, the event is activated again with the rising edge at the event input. The event is deactivated either

- by the falling edge at the event input (if an acknowledgment signal had previously come from the PLC with the Quit input or from the visualization) or
- by the rising edge at the Quit input (if a reset had previously been initiated by a falling edge at the event input).

If there is a reset of the event between event activation and the arrival of the acknowledgement, the next arrival of the event input is called "signal". A request is thus announced in case of already active events.

Example for the use of the ADSLOGEVENT function block in ST:

- Sending/acknowledging event logger signals from the PLC [▶ 137]

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC or CX (x86, x64, ARM)	Tc2_System (system)

3.4.2 ADSCLEAREVENTS

The function block sends and acknowledges messages to the TwinCAT EventLogger.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

The TwinCAT EventLogger was replaced by the successor TwinCAT 3 EventLogger. The older TwinCAT EventLogger is supported by TwinCAT 3 up to version 3.1.4024. Newer TwinCAT versions (>= 3.1.4026.0) only support the newer TwinCAT 3 EventLogger. PLC function blocks for this can be found in the PLC library Tc3_EventLogger.

Inputs

```
VAR_INPUT
    NETID    : T_AmsNetId;
    bClear   : BOOL;
    iMode    : UDINT;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Type	Description
NETID	T_AmsNetId	String containing the AMS network ID of the target device to which the ADS command is addressed (type: T_AmsNetId [▶ 122]).
bClear	BOOL	With the rising edge the events are deleted.
iMode	UDINT	Mode for deleting the events. Defined in the enum E_TcEventClearModes [▶ 119].
tTimeout	TIME	Indicates the time before the function is canceled.

Outputs

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

Name	Type	Description
bBusy	BOOL	When the function block is activated, this output is set to TRUE and remains set until feedback is received. As long as bBusy is TRUE, no new command can be executed.
bErr	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in iErrId. If the function block has a timeout error, bErr is TRUE and iErrId is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.
iErrId	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC or CX (x86, x64, ARM)	Tc2_System (system)

3.4.3 FB_SimpleAdsLogEvent

This function block allows the sending and acknowledgment of messages to the TwinCAT EventLogger. As opposed to the ADSLOGEVENT block, events cannot be parameterized from the PLC with the FB_SimpleAdsLogEvent block; however, events can be set, reset and acknowledged in a simple manner.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

The TwinCAT EventLogger was replaced by the successor TwinCAT 3 EventLogger. The older TwinCAT EventLogger is supported by TwinCAT 3 up to version 3.1.4024. Newer TwinCAT versions (>= 3.1.4026.0) only support the newer TwinCAT 3 EventLogger. PLC function blocks for this can be found in the PLC library Tc3_EventLogger.

Inputs

```
VAR_INPUT
    SourceId   : INT;
    EventId    : INT;
    bSetEvent  : BOOL;
    bQuit      : BOOL;
END_VAR
```

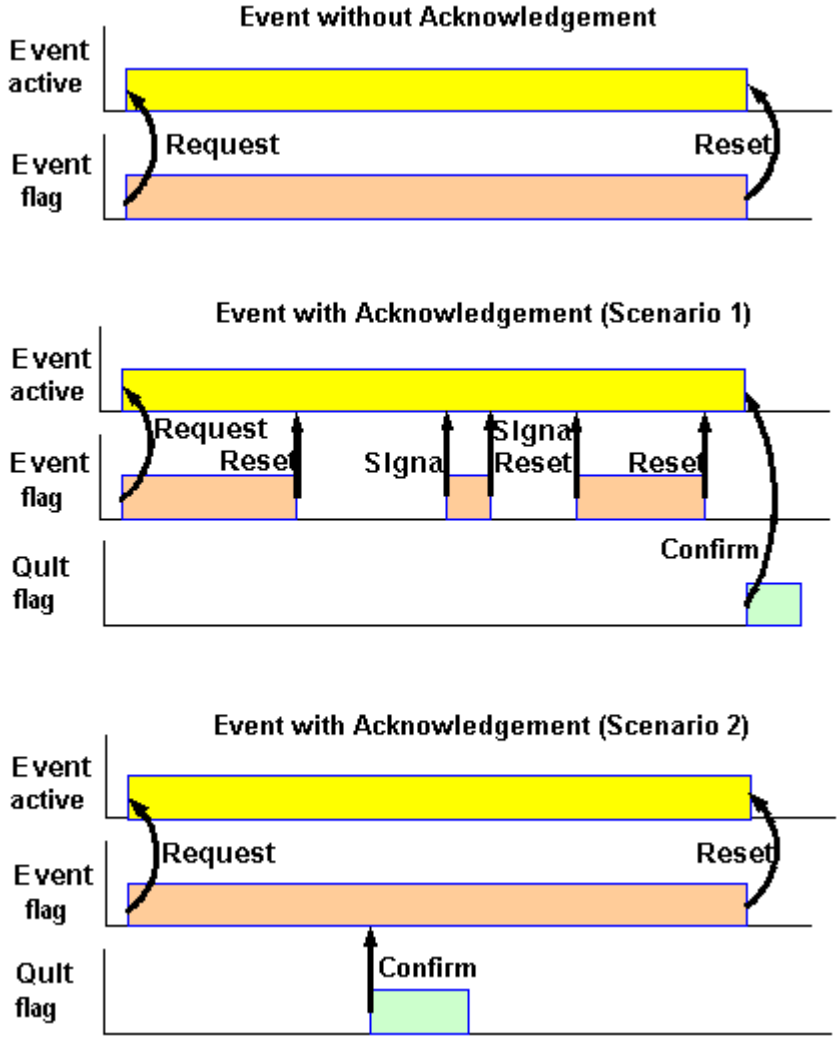
Name	Type	Description
SourceId	INT	ID of the source. Used to clearly identify the source in the EventLogger.
EventId	INT	ID of the event. Used to clearly identify the event in the EventLogger.
bSetEvent	BOOL	The "coming" of the event is signaled with the rising edge, the "going" of the event with the falling edge.
bQuit	BOOL	The event is acknowledged with the rising edge.

Outputs

```
VAR_OUTPUT
    ErrId     : UDINT;
    Error     : BOOL;
END_VAR
```

Name	Type	Description
ErrId	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.
Error	BOOL	This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in ErrId. If the function block has a timeout error, Error is TRUE and ErrId is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

Acknowledge messages



The upper figure represents the general sequence.

In the case of messages not requiring acknowledgment, the event is announced with the rising edge at the event input of the function block and is thus active in the EventLogger. The falling edge at the event input initiates the reset. This signal deletes the event in the EventLogger again.

In the case of messages requiring acknowledgment, the event is activated again with the rising edge at the event input. The event is deactivated either

- by the falling edge at the event input (if an acknowledgment signal had previously come from the PLC with the Quit input or from the visualization) or
- by the rising edge at the Quit input (if a reset had previously been initiated by a falling edge at the event input).

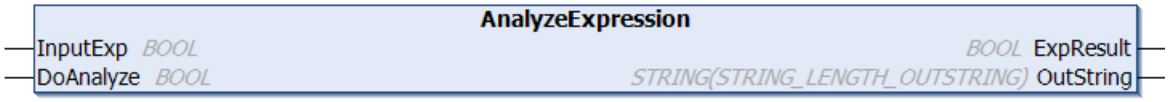
If there is a reset of the event between event activation and arrival of the acknowledgement, the next arrival of the event input is called "signal". A request is thus announced in case of already active events.

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC or CX (x86, x64, ARM)	Tc2_System (system)

3.5 IEC steps / SFC flags function blocks

3.5.1 AnalyzeExpression



The function block can be used in PLC projects that use the SFC flags. No instances are generated. The corresponding PLC library must be integrated in the project. Further configuration requirements can be found in the following explanatory notes.

The AnalyzeExpression and AnalyzeExpressionTable function blocks can be used to analyze transitions and step-enabling conditions. If a transition is not triggered after a set time, the transition can be analyzed using these function blocks.

The function blocks can only be used to analyze expressions or transitions that are implemented in the ST programming language.

- AnalyzeExpression:
  - The function block outputs the result of the analysis, i.e. the reason why no switch has occurred (that is, which partial condition(s) is/are not fulfilled), bundled in a STRING. The variables that form the partial conditions are linked to one another by operators (e.g. `bVar1 AND (bVar2 OR bVar3)`).
  - The SFC flag "SFCErrorAnalyzation" is used for the output.
- AnalyzeExpressionTable:
  - The function block outputs all non-switching variables individually. The individual variables are recorded or output as array elements. The information listed for each array element includes the name, address, comment and current value of the variable.
  - The SFC flag "SFCErrorAnalyzationTable" is used for the output.

Configuration requirements

The following settings are required to enable AnalyzeExpression or AnalyzeExpressionTable for SFC:

- Include the PLC library Tc2_System.
- Declare the following variable in the SFC-POU:
  `SFCEnableLimit: BOOL := TRUE;`
- In the Properties window, configure a maximum active time for the step(s) of the SFC diagram whose subsequent transition/switchover condition is to be analyzed (see also SFC element properties).
- Configure the SFC settings in the PLC project properties or in the POU properties (see also SFC flags and Command Properties (PLC project) > Category SFC):
  - **Flags** tab:
    Check the **Active** and **Declare** check boxes for the following SFC flags:
    SFCError, SFCEnableLimit, SFCErrorAnalyzation, SFCErrorAnalyzationTable
  - **Build** tab:
    Enable the option **Calculate active transitions only**.

Sample

The configurations mentioned above have been implemented in the following sample. For "Step1", the maximum active time was set to 1 s. If the associated outgoing transition "Trans_ST" has not triggered after 1 s, this transition is analyzed by the function blocks AnalyzeExpression and AnalyzeExpressionTable. The variable SFCError is set to TRUE and the variable SFCErrorStep is given the value 'Step 1'.

The analysis results "SFCErrorAnalyzation" or "SFCErrorAnalyzationTable" indicate which (partial) expression has not yet been triggered, so that "Step1" can be exited.

The transition "Trans_ST" consists of the expression

```
b1 AND (b2 OR b3);
```

- Situation 1: None of the three variables b1, b2, b3 is TRUE.
  - "SFCErrorAnalyzation" shows the analysis result 'b1 AND (b2 OR b3)'.
  - "SFCErrorAnalyzationTable" lists all three variables b1, b2, b3 with detailed variable information.
  - See also Figure 1.
- Situation 2: Variable b1 is set to TRUE. The analysis results change accordingly.
  - "SFCErrorAnalyzation" shows the analysis result '(b2 OR b3)'.
  - "SFCErrorAnalyzationTable" only lists the two variables b2 and b3 with the corresponding variable information.
  - See also Figure 2.

Figure 1:

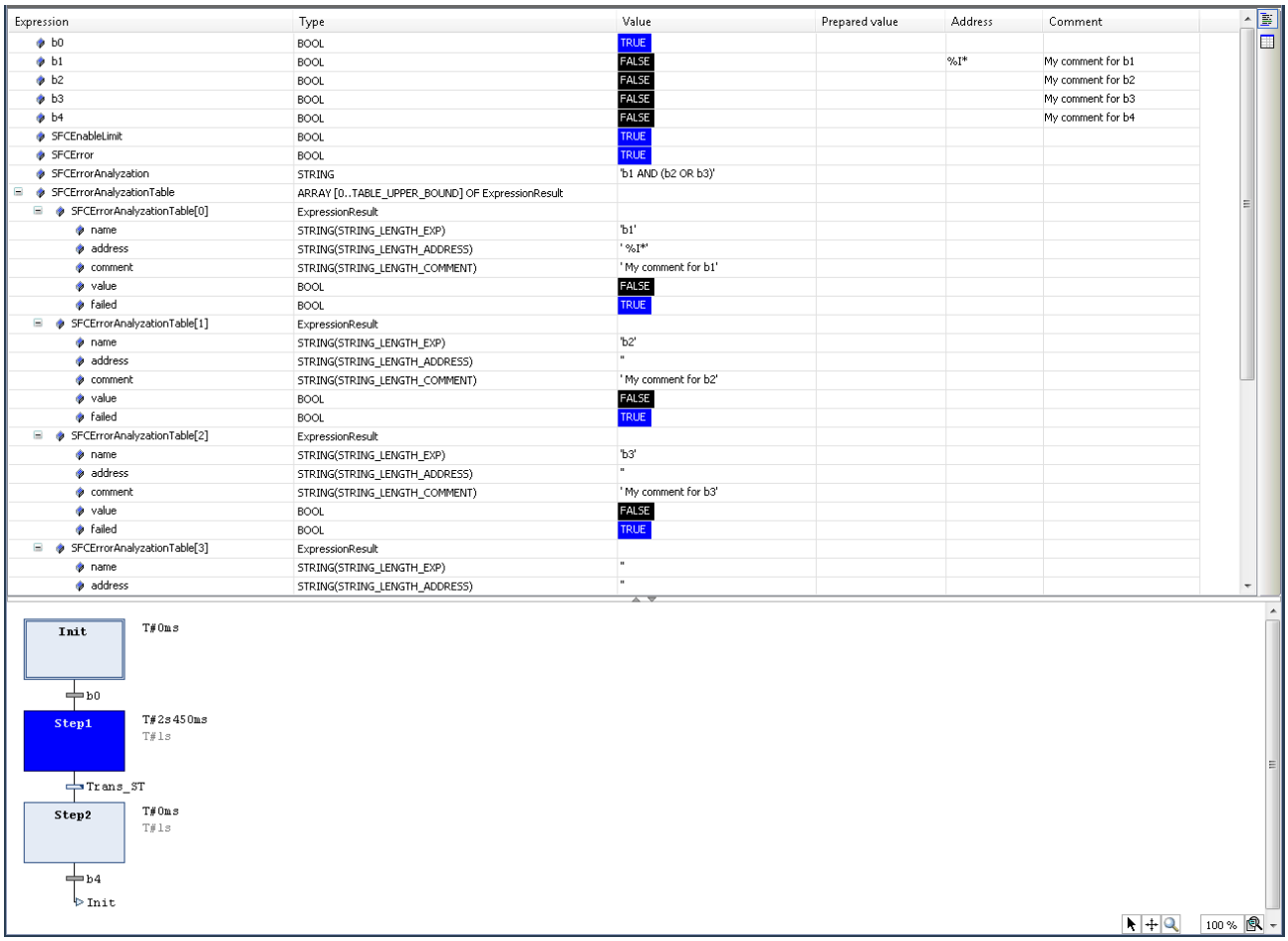
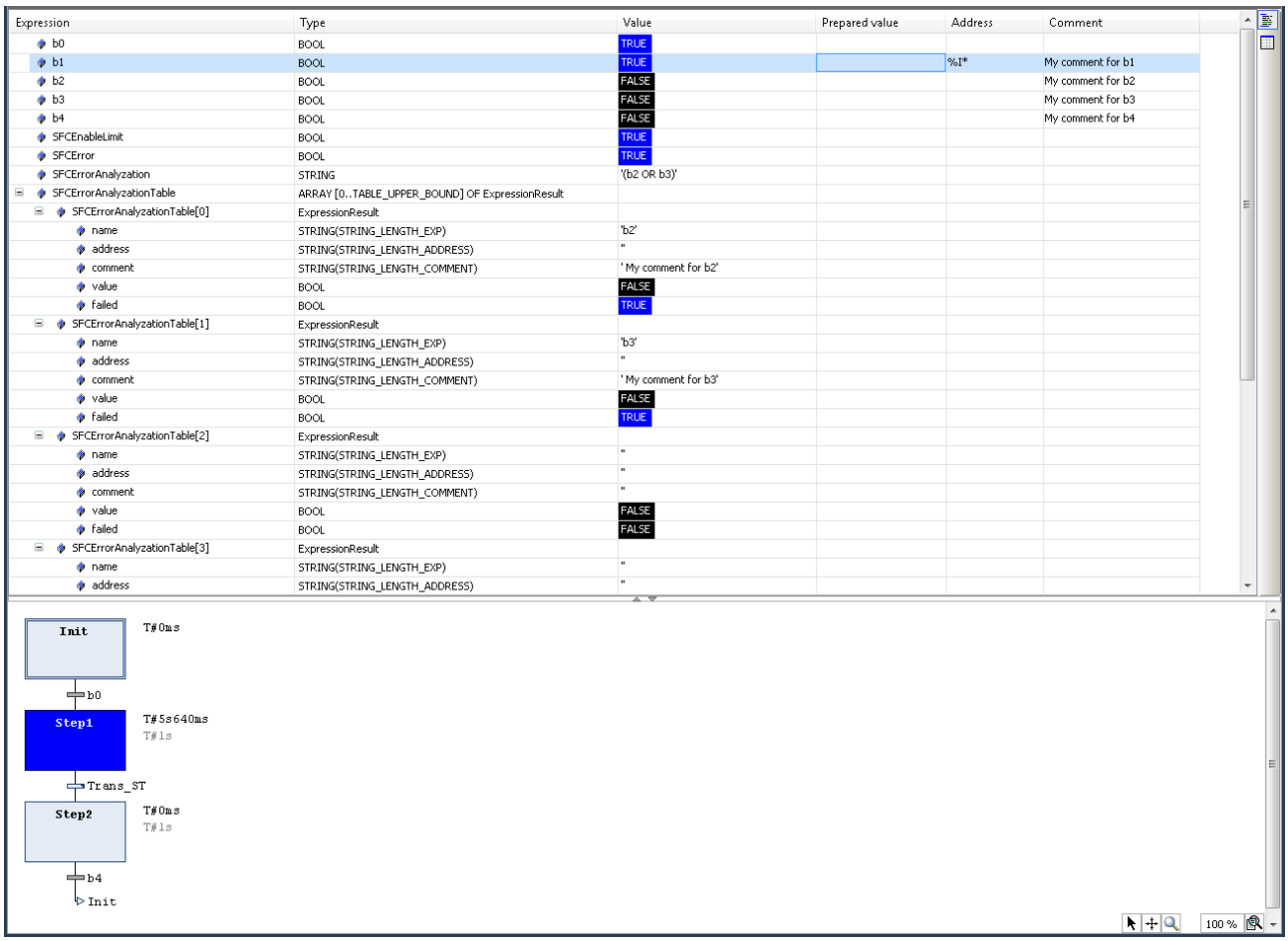


Figure 2:



Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.5.2 AnalyzeExpressionTable

The function block can be used in PLC projects that use the SFC flags. No instances are generated. The corresponding PLC library must be integrated in the project. Further configuration requirements can be found in the following explanatory notes.

The AnalyzeExpression and AnalyzeExpressionTable function blocks can be used to analyze transitions and step-enabling conditions. If a transition is not triggered after a set time, the transition can be analyzed using these function blocks.

The function blocks can only be used to analyze expressions or transitions that are implemented in the ST programming language.

- AnalyzeExpression:
  - The function block outputs the result of the analysis, i.e. the reason why no switch has occurred (that is, which partial condition(s) is/are not fulfilled), bundled in a STRING. The variables that form the partial conditions are linked to one another by operators (e.g. `bVar1 AND (bVar2 OR bVar3)`).
  - The SFC flag "SFCErrorAnalyzation" is used for the output.

- AnalyzeExpressionTable:
  - The function block outputs all non-switching variables individually. The individual variables are recorded or output as array elements. The information listed for each array element includes the name, address, comment and current value of the variable.
  - The SFC flag "SFCErrorAnalyzationTable" is used for the output.

Configuration requirements

The following settings are required to enable AnalyzeExpression or AnalyzeExpressionTable for SFC:

- Include the PLC library Tc2_System.
- Declare the following variable in the SFC-POU:
  `SFCEnableLimit: BOOL := TRUE;`
- In the Properties window, configure a maximum active time for the step(s) of the SFC diagram whose subsequent transition/switchover condition is to be analyzed (see also SFC element properties).
- Configure the SFC settings in the PLC project properties or in the POU properties (see also SFC flags and Command Properties (PLC project) > Category SFC):
  - **Flags** tab:
    Check the **Active** and **Declare** check boxes for the following SFC flags:
    SFCError, SFCEnableLimit, SFCErrorAnalyzation, SFCErrorAnalyzationTable
  - **Build** tab:
    Enable the option **Calculate active transitions only**.

Sample

The configurations mentioned above have been implemented in the following sample. For "Step1", the maximum active time was set to 1 s. If the associated outgoing transition "Trans_ST" has not triggered after 1 s, this transition is analyzed by the function blocks AnalyzeExpression and AnalyzeExpressionTable. The variable SFCError is set to TRUE and the variable SFCErrorStep is given the value 'Step 1'.

The analysis results "SFCErrorAnalyzation" or "SFCErrorAnalyzationTable" indicate which (partial) expression has not yet been triggered, so that "Step1" can be exited.

The transition "Trans_ST" consists of the expression

```
b1 AND (b2 OR b3);
```

- Situation 1: None of the three variables b1, b2, b3 is TRUE.
  - "SFCErrorAnalyzation" shows the analysis result 'b1 AND (b2 OR b3)'.
  - "SFCErrorAnalyzationTable" lists all three variables b1, b2, b3 with detailed variable information.
  - See also Figure 1.
- Situation 2: Variable b1 is set to TRUE. The analysis results change accordingly.
  - "SFCErrorAnalyzation" shows the analysis result '(b2 OR b3)'.
  - "SFCErrorAnalyzationTable" only lists the two variables b2 and b3 with the corresponding variable information.
  - See also Figure 2.

Figure 1:

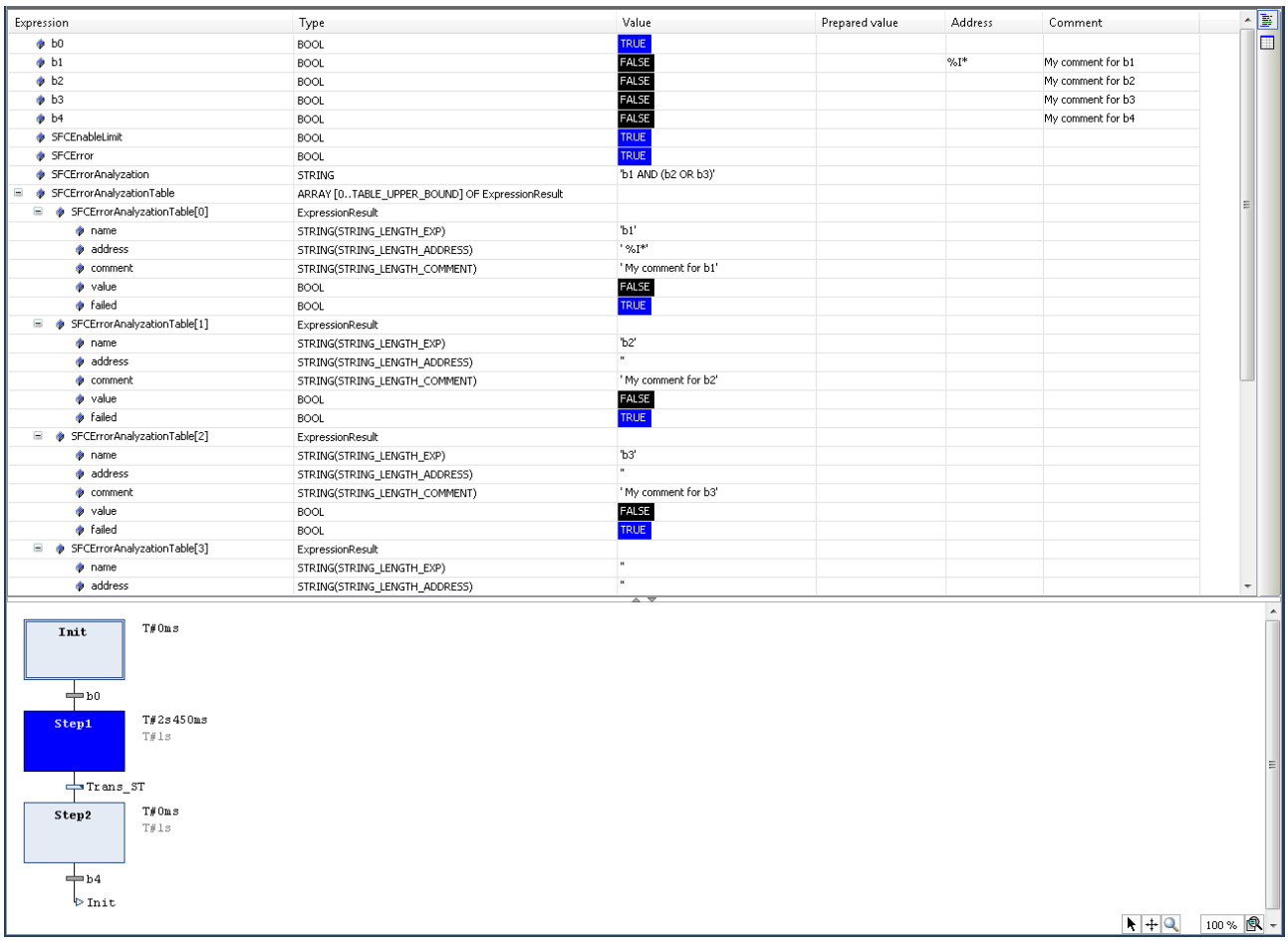
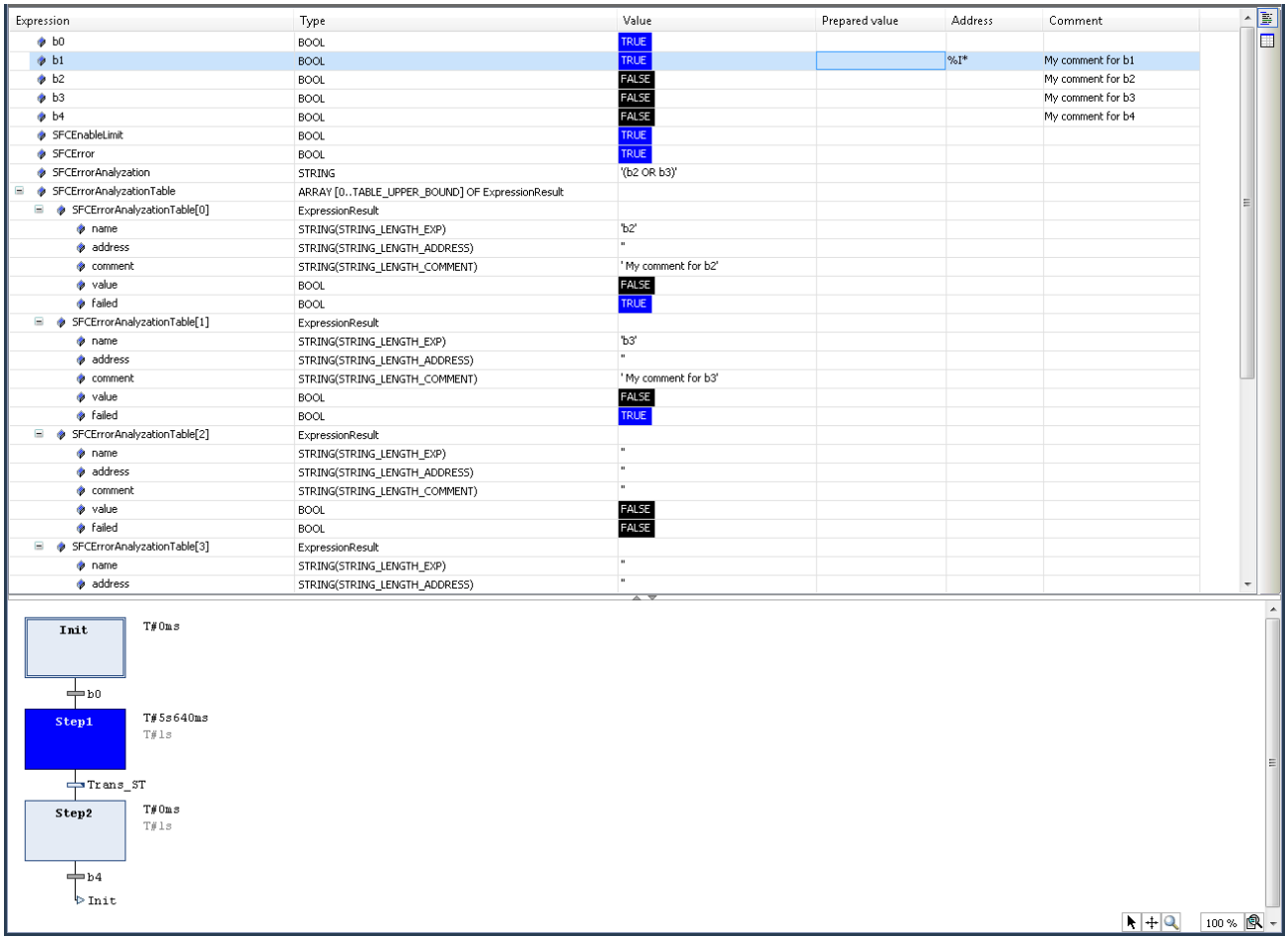

Expression
Type
Value
Prepared value
Address
Comment
b0 BOOL TRUE
b1 BOOL FALSE %I* My comment for b1
b2 BOOL FALSE My comment for b2
b3 BOOL FALSE My comment for b3
b4 BOOL FALSE My comment for b4
SFCEnableLimit BOOL TRUE
SFCError BOOL TRUE
SFCErrorAnalyzation STRING 'b1 AND (b2 OR b3)'
SFCErrorAnalyzationTable ARRAY [0..TABLE_UPPER_BOUND] OF ExpressionResult
SFCErrorAnalyzationTable[0] ExpressionResult
name STRING(STRING_LENGTH_EXP) 'b1'
address STRING(STRING_LENGTH_ADDRESS) '%I*'
comment STRING(STRING_LENGTH_COMMENT) ' My comment for b1'
value BOOL FALSE
failed BOOL TRUE
SFCErrorAnalyzationTable[1] ExpressionResult
name STRING(STRING_LENGTH_EXP) 'b2'
address STRING(STRING_LENGTH_ADDRESS) ''
comment STRING(STRING_LENGTH_COMMENT) ' My comment for b2'
value BOOL FALSE
failed BOOL TRUE
SFCErrorAnalyzationTable[2] ExpressionResult
name STRING(STRING_LENGTH_EXP) 'b3'
address STRING(STRING_LENGTH_ADDRESS) ''
comment STRING(STRING_LENGTH_COMMENT) ' My comment for b3'
value BOOL FALSE
failed BOOL TRUE
SFCErrorAnalyzationTable[3] ExpressionResult
name STRING(STRING_LENGTH_EXP) ''
address STRING(STRING_LENGTH_ADDRESS) ''
Init
T#0ms
b0
Step1
T#2s450ms
T#1s
Trans_ST
Step2
T#0ms
T#1s
b4
Init
100 %

Figure 2:


Expression
Type
Value
Prepared value
Address
Comment
b0 BOOL TRUE
b1 BOOL TRUE %I* My comment for b1
b2 BOOL FALSE My comment for b2
b3 BOOL FALSE My comment for b3
b4 BOOL FALSE My comment for b4
SFCEnableLimit BOOL TRUE
SFCError BOOL TRUE
SFCErrorAnalyzation STRING '(b2 OR b3)'
SFCErrorAnalyzationTable ARRAY [0..TABLE_UPPER_BOUND] OF ExpressionResult
SFCErrorAnalyzationTable[0] ExpressionResult
name STRING(STRING_LENGTH_EXP) 'b2'
address STRING(STRING_LENGTH_ADDRESS) ''
comment STRING(STRING_LENGTH_COMMENT) ' My comment for b2'
value BOOL FALSE
failed BOOL TRUE
SFCErrorAnalyzationTable[1] ExpressionResult
name STRING(STRING_LENGTH_EXP) 'b3'
address STRING(STRING_LENGTH_ADDRESS) ''
comment STRING(STRING_LENGTH_COMMENT) ' My comment for b3'
value BOOL FALSE
failed BOOL TRUE
SFCErrorAnalyzationTable[2] ExpressionResult
name STRING(STRING_LENGTH_EXP) ''
address STRING(STRING_LENGTH_ADDRESS) ''
comment STRING(STRING_LENGTH_COMMENT) ''
value BOOL FALSE
failed BOOL FALSE
SFCErrorAnalyzationTable[3] ExpressionResult
name STRING(STRING_LENGTH_EXP) ''
address STRING(STRING_LENGTH_ADDRESS) ''
Init
T#0ms
b0
Step1
T#5s640ms
T#1s
Trans_ST
Step2
T#0ms
T#1s
b4
Init
100 %

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

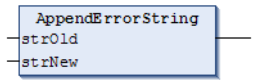
3.5.3 AnalyzeExpressionCombined

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	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.5.4 AppendErrorString

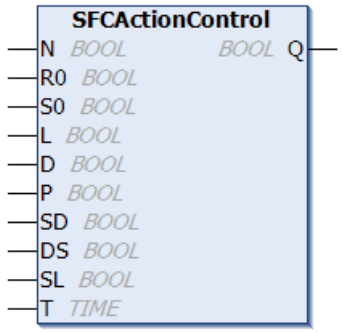


This function is required in PLC projects, which use the SFC flags. The function must not be called in the project. Only the corresponding PLC library must be included to the project.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.5.5 SFCActionControl



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

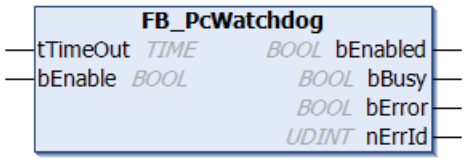
Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

3.6 Watchdog function blocks

3.6.1 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 activates a hardware watchdog on the PC. The watchdog is activated via bEnable = TRUE and the timeout time. The timeout time can range between 1 and 255 seconds. The watchdog is activated 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 deactivated via bEnable = FALSE or tTimeOut = 0.

NOTICE

PC reboot

The watchdog must be deactivated 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 activated, because otherwise the PC would reboot immediately once the timeout has elapsed.

Inputs

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

Name	Type	Description
tTimeOut	TIME	Watchdog time, after which a restart is performed.
bEnable	BOOL	Enabling/disabling the watchdog.

Outputs

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

Name	Type	Description
bEnabled	BOOL	TRUE = watchdog activated, FALSE = watchdog deactivated
bBusy	BOOL	This output remains TRUE until the function block has executed a command.
bError	BOOL	This output is switched to TRUE as soon as 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	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Example of calling the function block in ST:

```
PROGRAM MAIN
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 be integrated (category group)
TwinCAT v3.1.0	IPCs with the following mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051	PLC Lib Tc2_System

3.6.2 FB_PcWatchDog_BAPI

This functionality is only available on IPCs and Embedded PCs with a BIOS version which supports the BIOS-API.

The function block FB_PcWatchdog_BAPI activates a hardware watchdog on the PC. The watchdog is activated via bExecute = TRUE and the watchdog time. The watchdog time can range between 1 and 15300 seconds (255 minutes). The watchdog is activated via bEnable = TRUE and nWatchdogTimeS >=1 s.

Once the watchdog has been activated, the function block must be called cyclically at shorter intervals than nWatchdogTimeS, since the PC restarts automatically when nWatchdogTimeS 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.

NOTICE

PC reboot

The watchdog must be deactivated 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 activated, because otherwise the PC would reboot immediately once nWatchdogTimeS has elapsed.

Inputs

```
VAR_INPUT
    sNetID         : T_AmsNetID;
    nWatchdogTimeS : UDINT;
    bExecute       : BOOL;
    tTimeout       : TIME;
END_VAR
```

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the device (empty string or local network ID)
nWatchdogTimeS	UDINT	Watchdog time in seconds, 0 = deactivated, >0 (max. 15300) = activated.
bExecute	BOOL	The command is executed with a rising edge. The input must be reset as soon as the function block is no longer active (`bBusy=FALSE`).
tTimeout	TIME	Indicates the time until the internal ADS communication is terminated.

Outputs

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

Name	Type	Description
bEnabled	BOOL	TRUE = watchdog activated, FALSE = watchdog deactivated
bBusy	BOOL	This output remains TRUE until the function block has executed a command.
bError	BOOL	This output is switched to TRUE as soon as 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	UDINT	ADS error code [▶ 143] or command-specific error code of the last executed command. Is reset to 0 by the execution of a command at the inputs.

Sample of calling the function block in ST:

```
PROGRAM MAIN
VAR
    fbWatchdog     : FB_PcWatchdog_BAPI;
    nWatchdogTimeS : UDINT := 10;  (* 10s *)
    bEnabled       : BOOL; (* TRUE: watchdog is activated *)
    bError         : BOOL;
    nErrID         : UDINT;
    fbTimer        : TON := (IN := TRUE, PT := T#0S);
END_VAR
```

```
fbTimer();

(* 1st enable, then refresh watchdog every 1s *)
IF fbTimer.Q THEN
    fbWatchdog(
        sNetID         := '',
        nWatchdogTimeS := nWatchdogTimeS,
        bExecute       := TRUE,
        tTimeout       := T#5S,
    );

    IF NOT fbWatchdog.bBusy THEN
        bEnabled   := fbWatchdog.bEnabled;
        bError     := fbWatchdog.bError;
        nErrID     := fbWatchdog.nErrID;
```

```
        fbWatchdog(bExecute := FALSE);

        (* restart timer*)
        fbTimer(IN := FALSE);
        fbTimer(IN := TRUE, PT := T#1S); (* refresh watchdog every s *)
    END_IF
END_IF
```

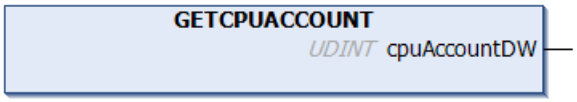
Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0	IPCs and Embedded PCs with a BIOS version which supports the BIOS-API.	PLC Lib Tc2_System-Version >=3.4.14.0

3.7 Time function blocks

3.7.1 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 100 ns ticks. The number is refreshed to a precision of 100 ns every time the PLC task is called, and can be used, for instance, for timing purposes. One unit is equivalent to 100 ns.

Inputs

```
VAR_INPUT
(*none*)
END_VAR
```

Outputs

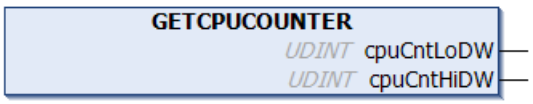
```
VAR_OUTPUT
    cpuAccountDW : UDINT;
END_VAR
```

Name	Type	Description
cpuAccountDW	UDINT	Current value of the PLC task ticker

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64)	Tc2_System (System)

3.7.2 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 CPUs internal clock rate, is output in a form converted into 100 ns ticks. The number is refreshed to a precision of 100 ns 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.

Inputs

```
VAR_INPUT
(*none*)
END_VAR
```

Outputs

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

Name	Type	Description
cpuCntLoDW	UDINT	Least significant 4 bytes of the count value
cpuCntHiDW	UDINT	Most significant 4 bytes of the count value

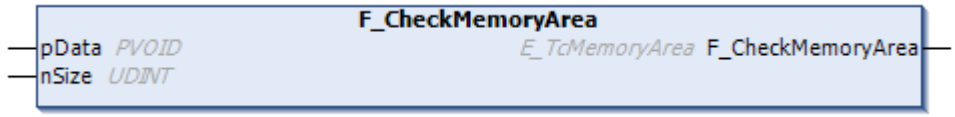
Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4 Functions

4.1 General functions

4.1.1 F_CheckMemoryArea



The function returns information about the memory area in which the requested variable with the specified size is located. A return value of type E_TcMemoryArea [▶ 120] is used for this purpose.

FUNCTION F_CheckMemoryArea: E_TcMemoryArea

Inputs

```
VAR_INPUT
    pData : PVOID;
    nSize : UDINT;
END_VAR
```

Name	Type	Description
pData	PVOID	Memory address of the variable
nSize	UDINT	Size of the variable in bytes

Example

```
PROGRAM MAIN
VAR
    nCounter : USINT;
    eMemAreaStatic : E_TcMemoryArea;
    pDynamicVariable : POINTER TO LREAL;
    eMemAreaDynamic : E_TcMemoryArea;
    pNull : PVOID := 0;
    eMemAreaUnknown : E_TcMemoryArea;
END_VAR
-----------------------------------------------------------------------

nCounter := nCounter + 1;
eMemAreaStatic := F_CheckMemoryArea( pData:=ADR(nCounter), nSize:=SIZEOF(nCounter) );

IF nCounter = 100 THEN
    pDynamicVariable := __NEW(LREAL);
    IF pDynamicVariable <> 0 THEN
        pDynamicVariable^ := 7 * 4.5;
        eMemAreaDynamic := F_CheckMemoryArea( pData:=pDynamicVariable, nSize:=SIZEOF(LREAL) );
        __DELETE(pDynamicVariable);
    END_IF
END_IF

eMemAreaUnknown := F_CheckMemoryArea( pData:=pNull, nSize:=1 );
```

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.4022	PC or CX (x86, x64, ARM)	Tc2_System (system)

4.1.2 F_CmpLibVersion

The function F_CmpLibVersion compares the version of the existing library with the required version. Each library has its own version information as a constant of type: ST_LibVersion. The name of the constant has the format: stLibVersion_libraryname.

FUNCTION F_CmpLibVersion: DINT

Inputs

```
VAR_INPUT
    stVersion   : ST_LibVersion;
    iMajor      : UINT;
    iMinor      : UINT;
    iBuild      : UINT;
    iRevision   : UINT;
END_VAR
```

Name	Type	Description
stVersion	ST_LibVersion	Version of the existing library (type: ST_LibVersion)
iMajor	UINT	Required major number
iMinor	UINT	Required minor number
iBuild	UINT	Required build number
iRevision	UINT	Required revision number

Return parameter	Version relationship
-1	Your version is lower than the required version.
0	Your version is the required version.
+1	Your version is higher than then required version.

Example in ST:

```
IF F_CmpLibVersion( stLibVersion_Tc2_System, 3, 3, 8, 0) >= 0 THEN
    (* newer lib ...*)
ELSE
    (* older lib... *)
END_IF
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.3 F_CreateIPv4Addr

The function generates a formatted (IPv4) Internet protocol network address and returns it as a return parameter of type string (e.g. 172.16.7.199).

FUNCTION F_CreateIPv4Addr : T_IPv4Addr

Inputs

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

Name	Type	Description
nIds	T_IPv4AddrArr	Byte array: Each byte corresponds to one address byte of the (IPv4) Internet Protocol network address. The address bytes have the network byte order (type: T_IPv4AddrArr [▶ 124]).

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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.4 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

Inputs

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

Name	Type	Description
sIPv4	T_IPv4AddrArr	Internet Protocol network address as a string (type: T_IPv4Addr [▶ 123]). E.g. 172.16.7.199.

Input parameters	Return parameter	Description
sIPv4 ≠ '' (empty string) and sIPv4 ≠ '0.0.0.0'	All bytes are null	Error during the conversion, check the formatting of the sIPv4 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.5 F_GetCpuCoreIndex

For a task index the function F_GetCpuCoreIndex returns the index of the CPU core on which the task runs.

If 0 is passed as task index, the CPU core index of the task in which the function is called is determined. If an invalid task index is passed, the function returns the CPU core index -1.

The function returns the determined CPU core index as a return parameter. It corresponds to the value of the **Core** column that is displayed in the **Real-time** sub-node below the SYSTEM node.

FUNCTION F_GetCpuCoreIndex: DINT

Inputs

```
VAR_INPUT
    nTaskIndex : DINT;
END_VAR
```

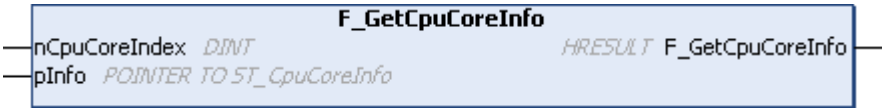
Name	Type	Description
nTaskIndex	DINT	Index of the task whose associated CPU index is to be determined. If 0 is passed as task index, the CPU core index of the task in which the function is called is determined.

See also:

- GETCURTASKINDEX [▶ 19]

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.11	PC or CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.24.0

4.1.6 F_GetCpuCoreInfo



The function F_GetCpuCoreInfo returns information about the CPU core whose index is passed to the function. The information that is read out includes the base time and the core limit for the specified CPU core.

The CPU core index to be passed can be determined with the function F_GetCpuCoreIndex [▶ 88].

The CPU core index corresponds to the value of the **Core** column that is displayed in the **Real-time** sub-node below the SYSTEM node. The information that can be read via the CPU core via the function F_GetCpuCoreInfo is also displayed in this view.

The function returns an error code as HRESULT (see also ADS Return Codes [▶ 143]). It indicates whether the function call was successful. If an invalid CPU core index is passed, the function returns an error (0x9811070B = invalid parameter values).

FUNCTION F_GetCpuCoreInfo: HRESULT

Inputs

```
VAR_INPUT
    nCpuCoreIndex : DINT;
    pInfo         : POINTER TO ST_CpuCoreInfo;
END_VAR
```

Name	Type	Description
nCpuCoreIndex	DINT	Index of the CPU core whose information is to be read.
pInfo	POINTER TO ST_CpuCoreInfo	Address of the variable that is to receive the read data. The address must point to an instance of type ST_CpuCoreInfo [▶ 121].

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.11	PC or CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.24.0

4.1.7 F_GetMappingPartner

The function F_GetMappingPartner returns the object ID (data type: OTCID) of the partner side of the mapping.

FUNCTION F_GetMappingPartner: OTCID

Inputs

```
VAR_INPUT
    p : PVOID;
    n : UDINT;
END_VAR
```

Name	Type	Description
P	PVOID	Memory address of the variable
n	UDINT	Size of the variable in bytes

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4020	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.8 F_GetMappingStatus

The function F_GetMappingStatus returns the current mapping status of a PLC variable. The function returns an ENUM value (data type: EPlcMappingStatus [▶ 120]) with the values MS_Unmapped, MS_Mapped or MS_Partial.

FUNCTION F_GetMappingStatus: EPlcMappingStatus

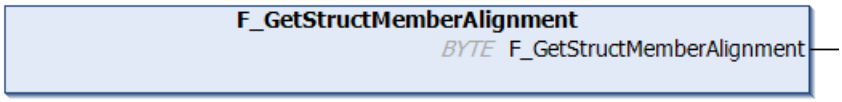
Inputs

```
VAR_INPUT
    p : PVOID;
    n : UDINT;
END_VAR
```

Name	Type	Description
P	PVOID	Memory address of the variable
n	UDINT	Size of the variable in bytes

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4020	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.9 F_GetStructMemberAlignment



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

FUNCTION F_GetStructMemberAlignment : BYTE

Inputs

```
VAR_INPUT
    (* keine Eingangsparameter *)
END_VAR
```

Return parameter	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 samples 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;
```

Alignment	SIZEOF(test4)	SIZEOF(test4.a1)	a1/a2 padding 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

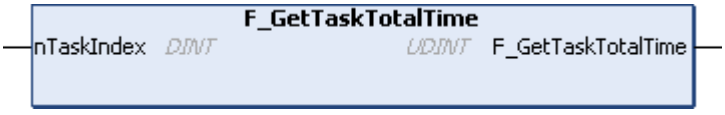
```
TYPE ST_D1
STRUCT
    ui16 : WORD := 16#1234;(* 34 12 *)
    ui8 : BYTE := 16#55;(* 55 *)
END_STRUCT
END_TYPE

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;
```

Alignment	SIZEOF(test5)	SIZEOF(test5.d1)	d1/d2 padding bytes	SIZEOF(test5.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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.10 F_GetTaskTotalTime



For a task index the function F_GetTaskTotalTime returns the total execution time of this task from the last cycle. The total execution time is the sum of the computing times of all modules that are registered on to the task.

If 0 is passed as task index, the value for the task in which the function is called is determined. If an invalid task index is passed, the function returns 0 as the total execution time.

The total execution time that was determined is shown as a multiple of 100 ns and returned by the function as a return parameter.

FUNCTION F_GetTaskTotalTime: UDINT

Inputs

```
VAR_INPUT
    nTaskIndex : DINT;
END_VAR
```

Name	Type	Description
nTaskIndex	DINT	Index of the task whose total execution time is to be determined. If 0 is passed as task index, the value for the task in which the function is called is determined.

See also:

- GETCURTASKINDEX [▶ 19]

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.11	PC or CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.24.0

4.1.11 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

Inputs

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

Name	Type	Description
sPathName	T_MaxString	Complete file name as string (type: T_MaxString [▶ 124]) in the form: 'X:\DIR\SUBDIR\FILENAME.EXT'.

Inputs/outputs

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

Name	Type	Description
sDrive	STRING	Drive ID (type: T_MaxString [▶ 124]) with a colon ('C:', 'A:' etc.)
sDir	T_MaxString	Directory name (type: T_MaxString [▶ 124]) including the leading and trailing backslash ('\BC \INCLUDE\', '\SOURCE\' etc.)
sFileName	T_MaxString	File name (type: T_MaxString [▶ 124])
sExt	T_MaxString	Contains the dot and the file extension (type: T_MaxString [▶ 124]) (example: '.C', '.EXE' etc.).

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

Sample of a call in ST:

The pathname: C:\TwinCAT\Plc\Project01\Data.txt is split into the following individual 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 type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.12 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

Inputs

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

Name	Type	Description
inVal32	DWORD	32-bit value to be changed.
bitNo	SINT	Number of the bit that is to be set (0-31). This number is internally converted to a modulo 32 value prior to execution.

Sample of calling the function in FBD:

Here bit 31 is set in the input value 0. The result is the (hex) value "80000000".

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.13 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.

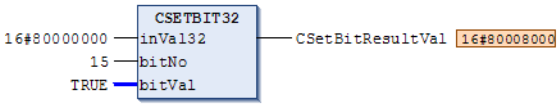
FUNCTION CSETBIT32 : DWORD

Inputs

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

Name	Type	Description
inVal32	DWORD	32-bit value
bitNo	SINT	Number of the bit to be set or reset (0-31). This number is internally converted to a modulo 32 value prior to execution.
bitVal	BOOL	Value to which the bit is to be set or reset (TRUE = 1, FALSE = 0).

Sample of calling the function in FBD:



Here bit 15 in the input value "16#80000000" is set to 1. The result (16#80008000) is assigned to the variable CSetBitResultVal.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.14 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.

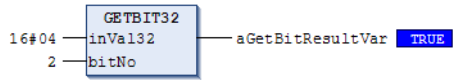
FUNCTION GETBIT32 : BOOL

Inputs

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

Name	Type	Description
inVal32	DWORD	32-bit value
bitNo	SINT	Number of the bit to be read (0-31). This number is internally converted to a modulo 32 value prior to execution.

Sample of calling the function in FBD:

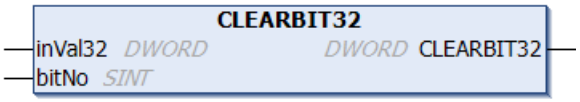


Here bit 2 in the input value "16#04" is queried and assigned to the boolean variable aGetBitResultVar. The query returns TRUE in this example.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.15 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.

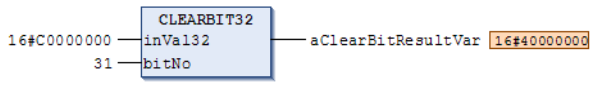
FUNCTION CLEARBIT32 : DWORD

Inputs

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

Name	Type	Description
inVal32	DWORD	32-bit value to be changed.
bitNo	SINT	Number of the bit that is to be set (0-31). This number is internally converted to a modulo 32 value prior to execution.

Sample of calling the function in FBD:

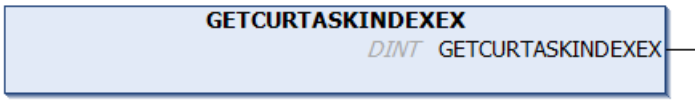


Here bit 31 in the input value "C0000000" is reset. The result is the (hex) value "40000000".

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.16 GETCURTASKINDEXEX



The function determines the task index of the task in which it is called. In contrast to the function block GETCURTASKINDEX [▶ 19], a distinction can be made as to whether the function is called in a cyclic real-time context or not.

FUNCTION GETCURTASKINDEXEX : DINT

Inputs

```
VAR_INPUT
    (*keine*)
END_VAR
```

Return parameter	Description
-1	The function is called from the Windows context.
0	The function is called from the real-time context, but not from a cyclic PLC task. This is the case, for example, with the automatic call of FB_init methods during initialization.
1 to n	The function is called from a cyclic PLC task. The return value is the task index.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.17 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.

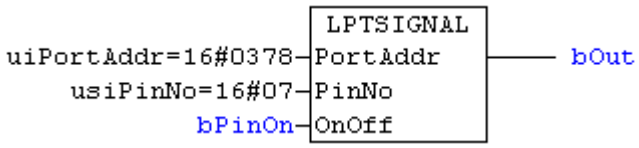
FUNCTION LPTSIGNAL: BOOL

Inputs

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

Name	Type	Description
PortAddr	UINT	Address of the port which is available for the desired LPT interface.
PinNo	INT	Number of the pin (pin 0 .. 7) to be written by the PLC.
OnOff	BOOL	State to be written to the pin.

Sample of calling the function in FBD:

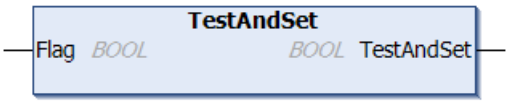


In the example, bit 7 of port 378 (hex) is set to 1.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.1.18 TestAndSet



You can use this function to check and set a flag. There is no option to interrupt the process. This allows data accesses to be synchronized. The mode of operation of a semaphore can be achieved with TestAndSet.

If the function call is successful, the function returns TRUE and the desired data can be accessed. If the function call is unsuccessful, the function returns FALSE and the desired data cannot be accessed. In this case, an alternative treatment must be provided for.

Inputs/outputs

```
VAR_IN_OUT
    Flag : BOOL; (* Flag to check if TRUE or FALSE *)
END_VAR
```

Name	Type	Description
Flag	BOOL	Boolean flag to be checked • if it was FALSE, the flag was free and is set (and therefore blocked from now on), and the function returns TRUE • if it was TRUE, the flag was already assigned (and therefore blocked), and the function returns FALSE

Sample

```
VAR_GLOBAL
    bGlobalTestFlag : BOOL;
END_VAR
```

```
VAR
    nLocalBlockedCounter : DINT;
END_VAR
```

```
IF TestAndSet(GVL.bGlobalTestFlag) THEN
    (* bGlobalTestFlag was FALSE, nobody was blocking, NOW
    bGlobalTestFlag is set to TRUE and blocking others *)

    (* ... *)

    (* remove blocking by resetting the flag *)
    GVL.bGlobalTestFlag := FALSE;
ELSE
    (* bGlobalTestFlag was TRUE, somebody is blocking *)
    nLocalBlockedCounter := nLocalBlockedCounter + 1;

    (* ... *)
END_IF
```

NEGATIVE sample

Caution is advised with a further encapsulation, e.g. in a function block, as this can destroy the desired atomic operation. Secure synchronization of data accesses can then no longer take place. In the following, a NEGATIVE sample is included that shows how the function may NOT be used. If two contexts were to request access at the same time in this implementation, both might assume that the access is allowed and simultaneous, unsecured accessing of the data would take place.

```
FUNCTION_BLOCK FB_MyGlobalLock
VAR_INPUT
    bLock    : BOOL; // set TRUE to lock & set FALSE to unlock
END_VAR
VAR_OUTPUT
    bLocked : BOOL;
END_VAR

IF bLock THEN
    TestAndSet(bLocked);
ELSE // unlock
    bLocked := FALSE;
END_IF
```

```
IF NOT GVL.fbGlobalLock.bLocked THEN
    GVL.fbGlobalLock(bLock := TRUE);

    (* ... *)

    GVL.fbGlobalLock(bLock := FALSE);
END_IF
```

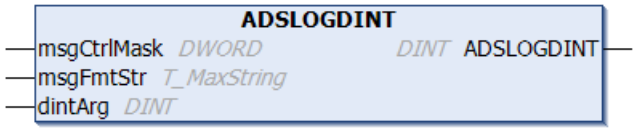
The function block FB_IecCriticalSection [▶ 14] offers the application of critical sections as an alternative Mutex method.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.2 ADS functions

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

FUNCTION ADSLOGDINT : DINT

Inputs

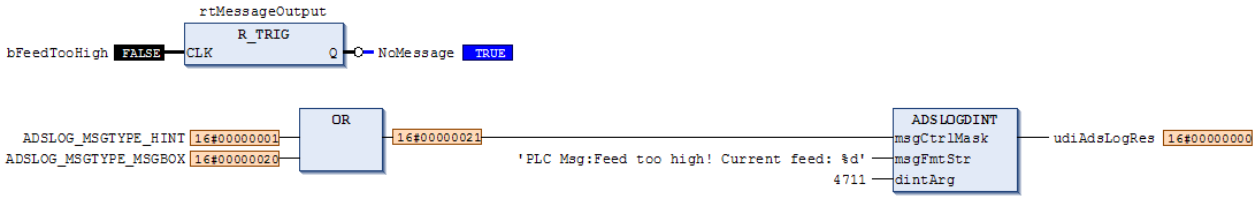
```
VAR_INPUT
    msgCtrlMask : DWORD;
    msgFmtStr   : T_MaxString;
    dintArg     : DINT;
END_VAR
```

Name	Type	Description
msgCtrlMask	DWORD	Control mask which determines the type and effect of the message output (see separate table).
msgFmtStr	T_MaxString	Contains the message to be issued (type: T_MaxString [▶ 124]). It can contain the formatting code %d for the output of a DINT value at any position.
dintArg	DINT	Contains the numerical value to be inserted into the message.

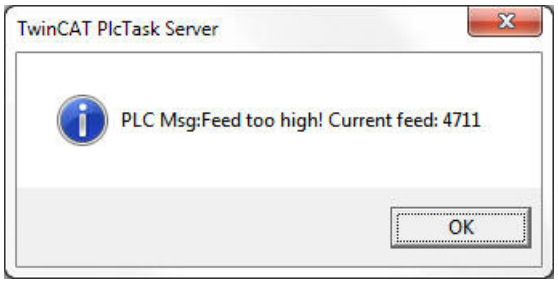
Constant	Description
ADSLOG_MSGTYPE_HINT	Message type is hint.
ADSLOG_MSGTYPE_WARN	Message type is warning.
ADSLOG_MSGTYPE_ERROR	Message type is error.
ADSLOG_MSGTYPE_LOG	Message is written into the log.
ADSLOG_MSGTYPE_MSGBOX	Message is output in a message box. Attention: This functionality is not available for Windows CE.
ADSLOG_MSGTYPE_STRING	Message is a directly given string (default).

The control masks can be ORed in the desired combination.

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:

```
PROGRAM MAIN
VAR
    rtMessageOutput: R_TRIG; (* Declaration *)
    bFeedTooHigh: BOOL;
    udiAdsLogRes: UDINT;
END_VAR

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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.2.2 ADSLOGLREAL

This function issues when called a message box holding a specifiable text on the screen, and writes an entry into the system's 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 stored format string must contain the characters %f at the desired location. Always remember that here too, as illustrated in the example, the function must be called using edge-control (see also the note in the description of ADSLOGDINT). The return value contains the function error code, or, if successful, 0.

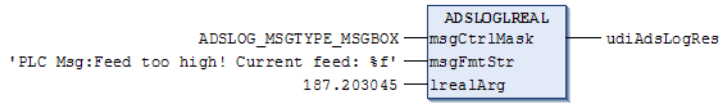
FUNCTION ADSLOGLREAL : DINT

Inputs

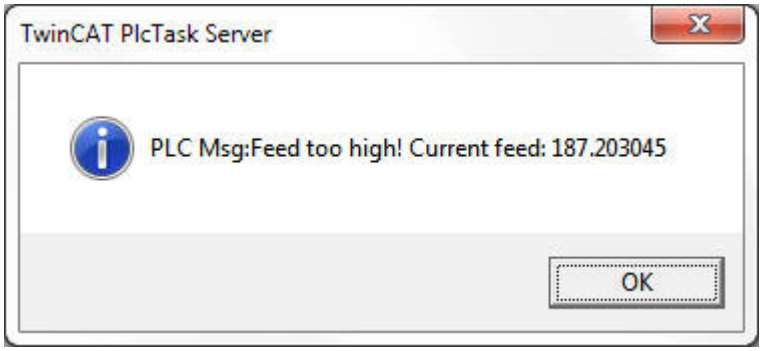
```
VAR_INPUT
    msgCtrlMask : DWORD;
    msgFmtStr   : T_MaxString;
    lrealArg    : LREAL;
END_VAR
```

Name	Type	Description
msgCtrlMask	DWORD	Control mask which determines the type and effect of the message output (see separate table). Listing of all the control masks for message output currently implemented as global constants in the library (see description of the function ADSLOGDINT [▶ 100]).
msgFmtStr	T_MaxString	Contains the message to be issued (type: T_MaxString [▶ 124]). It can contain the formatting code %d for the output of a DINT value at any position.
lrealArg	LREAL	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.

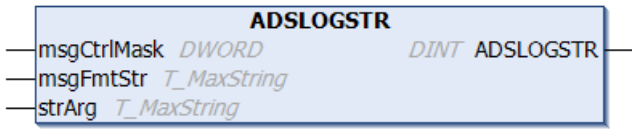
Example of calling the function in ST:

```
PROGRAM MAIN
VAR
    rtMessageOutput: R_TRIG; (* Declaration *)
    bTemperatureTooHigh: BOOL;
    udiAdsLogRes: UDINT;
END_VAR

rtMessageOutput(CLK := bTemperatureTooHigh);
IF rtMessageOutput.Q THEN
    udiAdsLogRes := ADSLOGLREAL( msgCtrlMask := ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_MSGBOX, msgFmt
Str := 'PLC Msg.: Max Temp. reached ! Temperature: %f', lrealArg := 187.203045);
END_IF;
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.2.3 ADSLOGSTR



This function issues when called a message box holding a specifiable text on the screen, and writes an entry into the system's log. In the text to be given out, a string (a sequence of characters) can be inserted in the text at a point specified by the user. For this purpose the stored format must contain the characters %s at the desired location. Always remember that here too, as illustrated in the example, the function must be called using edge-control (see also the note in the description of ADSLOGDINT [▶ 100]). The result value contains the function error code, or, if successful, 0.

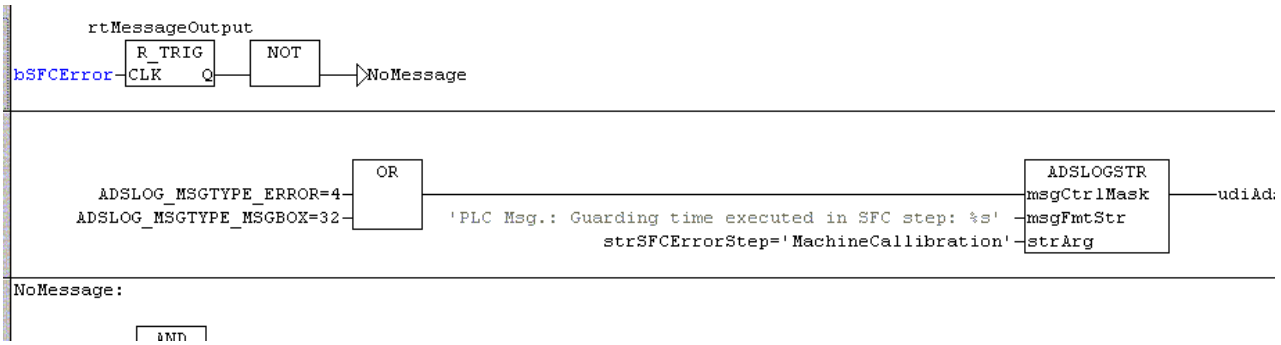
FUNCTION ADSLOGSTR : DINT

Inputs

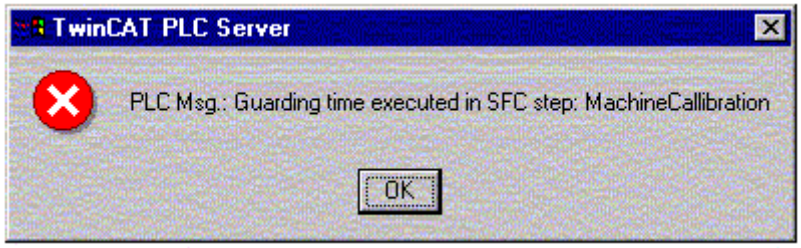
```
VAR_INPUT
    msgCtrlMask  : DWORD;
    msgFmtStr    : T_MaxString;
    strArg       : T_MaxString;
END_VAR
```

Name	Type	Description
msgCtrlMask	DWORD	Control mask which determines the type and effect of the message output (see separate table at ADSLOGDINT [▶ 100]).
msgFmtStr	T_MaxString	Contains the message to be issued (type: T_MaxString [▶ 124]). It can contain the formatting code %s for the output of a text argument at any position.
strArg	T_MaxString	Contains the string to be inserted in the message (type: T_MaxString [▶ 124]).

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:

```
PROGRAM MAIN
VAR
    strSFCErrorStep : STRING; (* Declaration*)
    rtMessageOutput: R_TRIG;
    bSFCError: BOOL;
END_VAR

rtMessageOutput(CLK := bSFCError);
IF rtMessageOutput.Q THEN
    udiAdsLogRes := ADSLOGSTR( msgCtrlMask := ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_MSGBOX, msgFmtS
tr := 'PLC Msg.: Guarding time executed in SFC step: %s', strArg := strSFCErrorStep);
END_IF;
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.2.4 F_CreateAmsNetId

The function generates a formatted NetID string (type: T_AmsNetID [▶ 122]) and returns it as return parameter (e.g. '127.16.17.3.1.1').

FUNCTION F_CreateAmsNetId : T_AmsNetId

Inputs

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

Name	Type	Description
nIds	T_AmsNetIdArr	Byte array (type: T_AmsNetIdArr [▶ 122]). Each byte corresponds to a number of the network address. The address bytes have a network byte order.

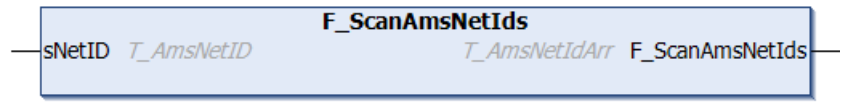
Example of a call in ST:

```
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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.2.5 F_ScanAmsNetIds



The function F_ScanAmsNetIds can be used to convert a string containing the TwinCAT network address to individual address bytes. The individual address bytes are converted from left to right and returned as an array of bytes (type: T_AmsNetIdArr [▶ 122]). The address bytes have a network byte order.

FUNCTION F_ScanAmsNetIds : T_AmsNetIdArr

Inputs

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

Name	Type	Description
sNetID	T_AmsNetID	TwinCAT network address as string (type: T_AmsNetId [▶ 122]). E.g. '127.16.17.3.1.1'

Input parameters	Return parameter	Description
sNetID ≠ '' (empty string) and sNetID ≠ '0.0.0.0.0.0'	All bytes are null	Error during the conversion, check the formatting of the sNetID string.

Example of a call in ST:

In the following example, a string with the network address '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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.3 Character functions

4.3.1 F_ToCHR

The function converts ASCII Code to STRING.

FUNCTION F_ToCHR: STRING

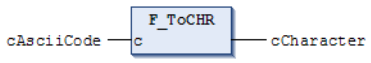
Inputs

```
VAR_INPUT
    c : BYTE;
END_VAR
```

Name	Type	Description
c	BYTE	ASCII code to be converted

Sample of calling the function in FBD:

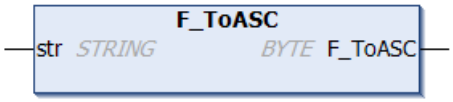
```
PROGRAM P_TEST
VAR
    sCharacter : STRING(1) := '';
    cAsciiCode : BYTE := 16#31;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.3.2 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

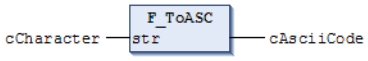
Inputs

```
VAR_INPUT
    str : STRING;
END_VAR
```

Name	Type	Description
str	STRING	String to convert

Sample of calling the function in FBD:

```
PROGRAM P_TEST
VAR
    sCharacter : STRING(1) := '1';
    cAsciiCode : BYTE := 0;
END_VAR
```

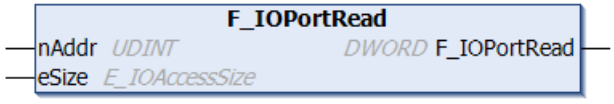


Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.4 I/O port access

4.4.1 F_IOPortRead



A digital I/O port is usually an I/O position with a width of 1 byte, which is mapped either in the memory or as a port. If a value is written at this point, the electrical signal at the output pins is modified according to the written bits. If a value is read from the input position, the current logical level at the input pins is returned as an individual bit value.

The function F_IOPortRead can be used to read an I/O position with a width of eSize. The function returns the read value as return parameter. See also description of the F_IOPortWrite [▶ 108] function.

FUNCTION F_IOPortRead : DWORD

Inputs

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

Name	Type	Description
nAddr	UDINT	I/O port address
eSize	E_IOAccessSize	Number of data bytes to be read (type: E_IOAccessSize [▶ 118])

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.4.2 F_IOPortWrite

The function F_IOPortWrite can be used to write to an I/O position with a width of eSize. See also description of the F_IOPortRead [▶ 107] function.

NOTICE

Hardware damage

A direct hardware access is not a problem, as long as data is only read. Write access can result in crashes and/or destroy the hardware or the data on the storage media. This function can damage the hardware to such an extent that it is no longer bootable.

FUNCTION F_IOPortWrite : BOOL

Inputs

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

Name	Type	Description
nAddr	UDINT	I/O port address
eSize	E_IOAccessSize	Number of data bytes to be written (type: E_IOAccessSize [▶ 118]).
nValue	DWORD	Value to be written.

Return parameter	Description
TRUE	No error
FALSE	Error

Sample code in ST:

In the following example a PLC Function block for direct control of a PC speaker is implemented with the aid of 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:

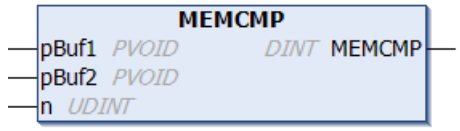
```
fbTrig( CLK := bExecute );
CASE nState OF
0:
IF fbTrig.Q THEN
    bBusy := TRUE;
    bError := FALSE;
    nErrID := 0;
    timer( IN := FALSE );

    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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.5 Memory functions

4.5.1 MEMCMP



The function MEMCMP allows the values of PLC variables in two different memory areas to be compared.

MEMCMP FUNCTION: DINT

Inputs

```
VAR_INPUT
    pBuf1 : PVOID;
    pBuf2 : PVOID;
    n     : UDINT;
END_VAR
```

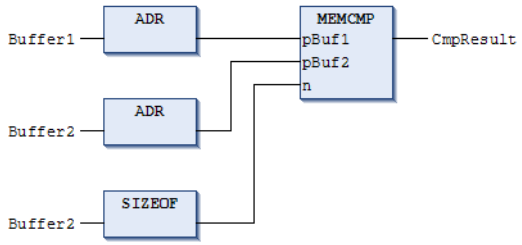
Name	Type	Description
pBuf1	PVOID	Start address of the first memory area (the first data buffer).
pBuf2	PVOID	Start address of the second memory area (the second data buffer).
n	UDINT	Number of bytes to be compared.

The function compares the first n bytes in the two data buffers and returns a value that corresponds to their ratio.

Return parameter	Ratio of the first byte that is different in the first and second data buffers
-1	pBuf1 lower 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:

```
PROGRAM MAIN
VAR
    Buffer1   : ARRAY[0..3] OF BYTE;
    Buffer2   : ARRAY[0..3] OF BYTE;
    CmpResult : DINT;
END_VAR
```

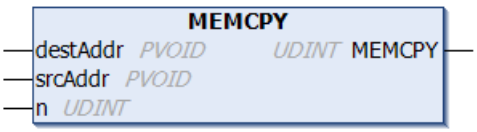


In this example, 4 bytes of data in Buffer2 are compared with those in Buffer1. The first differing byte is larger in Buffer1 than it is in Buffer2.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.5.2 MEMCPY



The function MEMCPY can be used to copy the values of PLC variables from one memory area to another.

NOTICE

System crash or access to unauthorized memory areas

Since the function directly accesses the physical memory, special care must be taken when using it. Incorrect parameter values can result in a system crash, or in access to forbidden memory areas.

The behavior of MEMCPY is undefined if the destination and source memory areas overlap. This is the case, for example, when several values stored in an array are to be moved one position forward or backward. In such a case, use the function MEMMOVE [▶ 112].

FUNCTION MEMCPY : UDINT

Inputs

```
VAR_INPUT
    destAddr : PVOID;
    srcAddr  : PVOID;
    n        : UDINT;
END_VAR
```

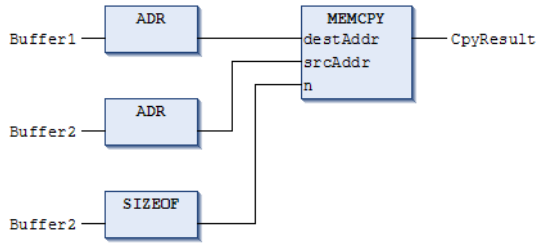
Name	Type	Description
destAddr	PVOID	Start address of the destination memory area.
srcAddr	PVOID	Start address of the source memory area.
n	UDINT	Number of bytes to be copied.

The function copies n bytes from the memory area with the start address srcAddr to the memory area with the start address 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:

```
PROGRAM MAIN
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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.5.3 MEMMOVE

Use the MEMMOVE function to copy values from one memory area to another; the memory areas may overlap.

NOTICE

System crash or access to unauthorized memory areas

Since the function directly accesses the physical memory, special care must be taken when using it. Incorrect parameter values can result in a system crash, or in access to forbidden memory areas.

FUNCTION MEMMOVE : UDINT

Inputs

```
VAR_INPUT
    destAddr : PVOID;
    srcAddr  : PVOID;
    n        : UDINT;
END_VAR
```

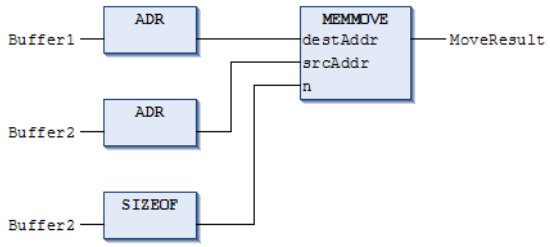
Name	Type	Description
destAddr	PVOID	Start address of the destination memory area.
srcAddr	PVOID	Start address of the source memory area.
n	UDINT	Number of bytes to be copied.

The function copies n bytes from the memory area with the start address srcAddr to the memory area with the start address 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:

```
PROGRAM MAIN
VAR
    Buffer1    : ARRAY[0..3] OF BYTE;
    Buffer2    : ARRAY[0..3] OF BYTE;
    MoveResult : UDINT;
END_VAR
```

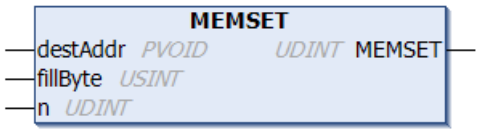


In the example, 4 bytes are moved from Buffer2 to Buffer1.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.5.4 MEMSET



The function MEMSET allows PLC variables in a particular memory area to be set to a particular value.

NOTICE

System crash or access to unauthorized memory areas

Since the function directly accesses the physical memory, special care must be taken when using it. Incorrect parameter values can result in a system crash, or in access to forbidden memory areas.

MEMSET FUNCTION: UDINT

Inputs

```
VAR_INPUT
    destAddr : PVOID;
    fillByte : USINT;
    n        : UDINT;
END_VAR
```

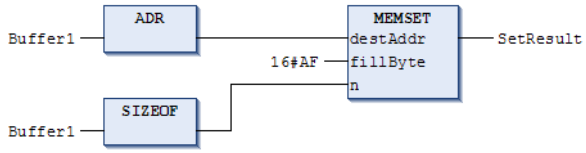
Name	Type	Description
destAddr	PVOID	Start address of the memory area that is to be set.
fillByte	USINT	Value of the filler byte.
n	UDINT	Number of bytes to be set.

The function fills n bytes with the fillByte values; starting from the memory area with the start address destAddr.

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:

```
PROGRAM MAIN
VAR
    Buffer1 : ARRAY[0..3] OF BYTE;
    SetResult : UDINT;
END_VAR
```



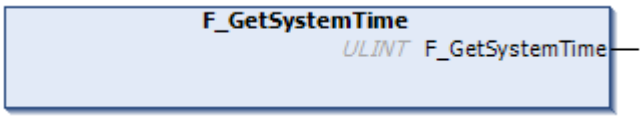
In the example, 4 bytes in Buffer1 are set to the value 0xAF.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.6 Time functions

4.6.1 F_GetSystemTime



This function can be used to read the operating system time stamp. 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 utilized for timing tasks or time measurements. One unit corresponds to 100 ns. The time represents the number of 100 ns intervals since 1 January 1601.

Outputs

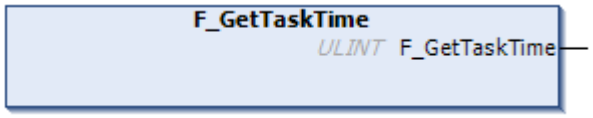
```
VAR_OUTPUT
    F_GetSystemTime   : ULINT;
END_VAR
```

The return value contains the timestamp.

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.17.0

4.6.2 F_GetTaskTime



This function can be used to read the start time of the task (time at which the task should start). The function always returns the start time of the task in which the function was called. The time stamp is a 64-bit integer value with an accuracy of 100 ns. It can be used for timing tasks or time measurements, among other things. One unit corresponds to 100 ns. The time represents the number of 100 ns intervals since 1 January 1601.

Outputs

```
VAR_OUTPUT
    F_GetTaskTime : ULINT;
END_VAR
```

The return value contains the timestamp.

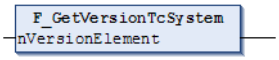
Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.17.0

4.7 [Obsolete]

4.7.1 F_GetVersionTcSystem

This function is obsolete and should not be used. Use the global constant stLibVersion_Tc2_System [▶ 131] instead to read version information of the PLC library.



This function can be used to read PLC library version information.

FUNCTION F_GetVersionTcSystem : UINT

Inputs

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

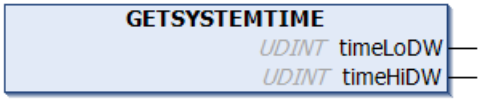
Name	Type	Description
nVersionElement	INT	**nVersionElement** : Version element to be read. Possible parameters: • 1 : major number; • 2 : minor number; • 3 : revision number;

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.7.2 GETSYSTEMTIME

This function block is replaced by the newer function F_GetSystemTime(), which only needs one return value, not two.



With this function 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 utilized for timing tasks or time measurements. One unit corresponds to 100 ns. The time represents the number of 100 ns intervals since 1 January 1601.

See: F_GetSystemTime [▶ 114]

Inputs

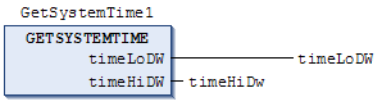
```
VAR_INPUT
(*none*)
END_VAR
```

Outputs

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

Name	Type	Beschreibung
timeLoDW	UDINT	Contains the low-value 4 bytes of the time stamp.
timeHiDW	UDINT	Contains the high-value 4 bytes of the time stamp.

Sample of calling the function block in FBD:



The sample illustrates calling the function 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

4.7.3 GETTASKTIME

This function block is replaced by the newer function F_GetTaskTime(), which only needs one return value, not two.

This function block can be used to read the start time of the task (time at which the task should start). The function block always returns the start time of the task in which the function block instance was called. The time stamp is a 64-bit integer value with an accuracy of 100 ns. It can be used for timing tasks or time measurements, among other things. One unit corresponds to 100 ns. The time represents the number of 100 ns intervals since 1 January 1601.

See: F_GetTaskTime [▶ 115]

Inputs

```
VAR_INPUT
(*none*)
END_VAR
```

Outputs

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

Name	Type	Beschreibung
timeLoDW	UDINT	Contains the low-value 4 bytes of the time stamp.
timeHiDW	UDINT	Contains the high-value 4 bytes of the time stamp.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5 Data types

5.1 E_IOAccessSize

Byte size of I/O position (number of bytes to be written or read).

```
TYPE E_IOAccessSize :
(
    NoOfByte_Byte  := 1,
    NoOfByte_Word  := 2,
    NoOfByte_DWord := 4
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.2 E_OpenPath

The variable of this type selects generic or one of the TwinCAT system paths on the target device to perform the file open operation.

```
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
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.3 E_SeekOrigin

A variable of this type shows the origin point by moving the pointer.

```
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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.4 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.5 E_TcEventClearModes

```
TYPE E_TcEventClearModes :
(
    TCEVENTLOGIOFFS_CLEARACTIVE := 1,
    TCEVENTLOGIOFFS_CLEARLOGGED ,
    TCEVENTLOGIOFFS_CLEARALL
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.6 E_TcEventPriority

```
TYPE E_TcEventPriority :
(
    TCEVENTPRIO_IMPLICIT := 0
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.7 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 (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.8 E_TcMemoryArea

The F_CheckMemoryArea [▶ 85] function returns information about the memory area in which the requested variable with the specified size is located. A return value of type E_TcMemoryArea is used for this purpose.

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_TcMemoryArea :
(
    Unknown := 0,
    Static  := 1, // static PLC memory
    Dynamic := 2, // dynamic memory
    CNC     := 3
)UDINT;
END_TYPE
```

Name	Description
Unknown	The memory area is unknown. For example, this could be memory in a Windows context. The memory area is also output as unknown if the specified memory size results in two different memory areas being involved. Furthermore, the memory area is output as unknown if it is a stack memory.
Static	These are static PLC memories.
Dynamic	These are dynamically allocated memories, which were allocated during the runtime or during the initialization phase of the PLC.
CNC	These are memories of the CNC driver.

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.4022	PC or CX (x86, x64, ARM)	Tc2_System (system)

5.9 E_UsrLED_Color

```
TYPE E_UsrLED_Color :
(
    eUsrLED_Off   := 0,
    eUsrLED_Red   := 1,
    eUsrLED_Blue  := 2,
    eUsrLED_Green := 3
);
END_TYPE
```

5.10 EPlcMappingStatus

```
Type EPlcMappingStatus :
(
    MS_Unmapped,
    MS_Mapped,
    MS_Partial
);
End_TYPE
```

This data type is defined in the global type system.

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4020	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.11 ST_AmsAddr

A variable of this type contains the TwinCAT network address.

```
TYPE ST_AmsAddr :
STRUCT
    netId   : T_AmsNetIdArr;
    port    : T_AmsPort;
END_STRUCT
END_TYPE
```

Name	Type	Description
netId	T_AmsNetIdArr	AMS network address (type: T_AmsNetIdArr [▶ 122])
port	T_AmsPort	AMS port number (type: T_AmsPort [▶ 122])

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.12 ST_CpuCoreInfo

A variable of this type contains information about a CPU core. The information can be read for a particular CPU core with the aid of the function F_GetCpuCoreInfo [▶ 88] and the corresponding CPU core index.

```
TYPE ST_CpuCoreInfo :
STRUCT
    bRTCore       : BOOL;
    bIsolatedCore : BOOL;
    nBaseTime     : UDINT;
    nCoreLimit    : UDINT;
END_STRUCT
END_TYPE
```

Name	Type	Description
bRTCore	BOOL	This variable has a value of TRUE if it is a real-time kernel.
bIsolatedCore	BOOL	This variable has a value of TRUE if it is an isolated core.
nBaseTime	UDINT	Base time of the CPU core, specified as a multiple of 100 ns
nCoreLimit	UDINT	Core limit of the CPU core, specified in %

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.11	PC or CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.24.0

5.13 SYSTEMINFOTYPE

This TwinCAT2 data type does not exist in TwinCAT 3 any more.

SystemInfoType is replaced by PlcAppSystemInfo which is a Global Data Type.

5.14 SYSTEMTASKINFOTYPE

This TwinCAT2 data type does not exist in TwinCAT 3 any more.

SystemTaskInfoType is replaced by PlcTaskSystemInfo which is a Global Data Type.

5.15 T_AmsNetID

A PLC variable of this type is a string containing the AMS network ID of the target device to which the ADS command is directed. The string consists of six numerical fields, separated by dots. 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 ID of the local device is automatically assumed.

Namespace: Tc2_System
Library: Tc2_System (Tc2_System.compiled-library)

```
TYPE T_AmsNetID : STRING(23);
END_TYPE
```

5.16 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 [▶ 105]

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.17 T_AmsPort

A variable of this type contains the ADS port number. 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.

```
TYPE T_AmsPort : UINT;
END_TYPE
```

Table with specified ADS port numbers:

ADS device	Port number
Cam controller	900
Runtime system 1 Runtime system 2 Runtime system 3 Runtime system 4 Runtime system 5 Runtime system n	Runtime system 1: 851 (in TwinCAT 2: 801) Runtime system 2: 852 (in TwinCAT 2: 811) Runtime system 3: 853 (in TwinCAT 2: 821) Runtime system 4: 854 (in TwinCAT 2: 831) Runtime system 5: 855 Runtime system n: 850 + n, and so on
NC	500
Reserved	400
I/O	300
Real-time core	200
Event System (logger)	100

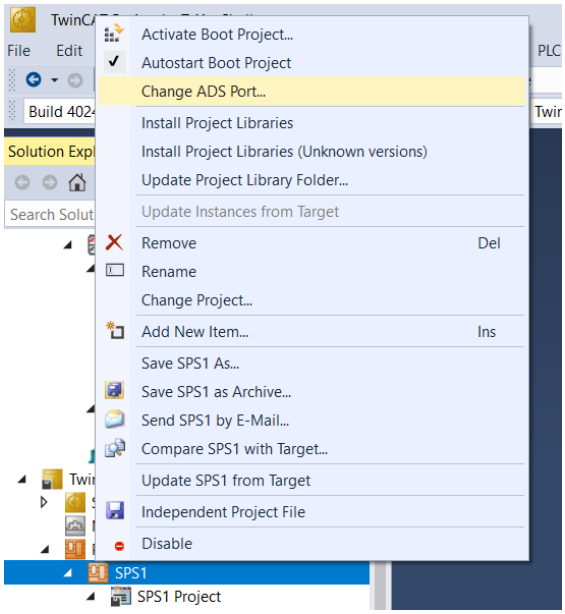
Up to four independent PLC runtime systems (PLC projects) can run on a TwinCAT 2 system; each PLC project is to be regarded as a stand-alone PLC. The port number and the network address are both required as input parameters when the ADS blocks are called.

The ADS ports from 800 to 899 are generally available for the PLC in TwinCAT3. In order to separate TwinCAT 2 and TwinCAT 3 systems, however, we recommend using only the ports from 851 to 899.

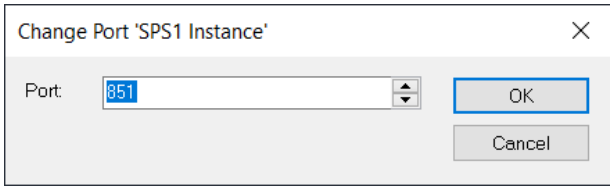
If you set the ADS port with the help of the dialog, port 851 is displayed as the lowest port that can be set. In order to use the range 800-850, you need to type in the port number.

Proceed as follows to enter the port number via the dialog box:

1. Right-click the desired PLC project.
2. Click **Change ADS Port**.



⇨ The dialog box opens.



3. Select the desired port number using the arrow keys or type in the desired port number.
4. Confirm the entry with **OK**.

⇨ The port number has been entered in the system.

To ensure separation between TwinCAT 2 and TwinCAT 3 systems, we recommend using only the ADS ports from 851 to 899.

ADS ports outside the range from 800 bis 899 will not be accepted by the input system.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.18 T_IPv4Addr

A variable of this type is a string with the (Ipv4) Internet protocol network address. E.g. '172.16.7.199'.

```
TYPE T_IPv4Addr : STRING(15);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.19 T_IPv4AddrArr

The variable of this type is a array of bytes containing the (IPv4) Internet Protocol network address.

```
TYPE T_IPv4AddrArr: ARRAY[0..3] OF BYTE;
END_TYPE
```

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 [▶ 87]

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.20 T_MaxString

The variable of this type is PLC string with the maximal length. Longer strings are allowed, but the string functions are limited to 255 characters.

```
TYPE T_MaxString : STRING(MAX_STRING_LENGTH);
END_TYPE
```

```
VAR_GLOBAL CONSTANT
    MAX_STRING_LENGTH : UDINT := 255;
ENd_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

5.21 TcEvent

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

The TwinCAT EventLogger was replaced by the successor TwinCAT 3 EventLogger. The older TwinCAT EventLogger is supported by TwinCAT 3 up to version 3.1.4024. Newer TwinCAT versions (>= 3.1.4026.0) only support the newer TwinCAT 3 EventLogger. PLC function blocks for this can be found in the PLC library Tc3_EventLogger.

```
TYPE TcEvent
STRUCT
    Class               : UDINT;
    Prio                : UDINT;
    Id                  : UDINT;
    bQuitRequired       : BOOL;
```

```
    DataFormatStrAddress : PVOID;
    UserFlags            : DWORD;
    Flags                : DWORD;
    StreamType           : UDINT;
    SourceString         : STRING[15]; (* TCEVENT_SRCNAMESIZE *)
    SourceId             : UDINT;
    ProgId               : STRING[31]; (* TCEVENT_FMTPRGSIZE *)
END_STRUCT
END_TYPE
```

Name	Type	Description
Class	UDINT	Event class, get value from enum E_TcEventClass [▶ 119].
Prio	UDINT	Priority of the event within a class, freely selectable number (1..MaxUDINT)
Id	UDINT	Id of the event, used for unique identification in the EventLogger.
bQuitRequired	BOOL	Flag for switching the acknowledgement requirement on and off (TRUE → acknowledgement required)
DataFormatStrAddress	PVOID	Address of a string, string contains formatting instructions (e.g. %d%f formats an integer and a real (float) value).
UserFlags	DWORD	32-bit number for free disposal
Flags	DWORD	32-bit number identifying the event, the meaning of the individual bits are declared in the global variables [▶ 126] of the library.
StreamType	UDINT	Type of the event, get value from the enum E_TcEventStreamType [▶ 119].
SourceString	STRING	String with the source name (max. 15 characters [▶ 126])
SourceId	UDINT	Source-ID
ProgId	STRING	String (Prog-Id) with the name of the formatter (max. 31 characters [▶ 126]). Default: 'TcEventLogger.TcLogFormatter' or 'TcEventFormatter.TcXmlFormatter'

Requirements

Development environment	Target platform	PLC libraries to be integrated (category group)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC or CX (x86, x64, ARM)	Tc2_System (system)

6 Global constants

6.1 Constants

Port numbers

Port numbers	Value	Description
AMSPORT_LOGGER	100	Port number of the Standard-Logger.
AMSPORT_EVENTLOG	110	Port number of the TwinCAT Eventlogger.
AMSPORT_R0_RTIME	200	Port number of the TwinCAT Realtime Server.
AMSPORT_R0_IO	300	Port number of the TwinCAT I/O Server.
AMSPORT_R0_NC	500	Port number of the TwinCAT NC Server.
AMSPORT_R0_NCSAF	501	Port number of the TwinCAT NC Serves (Task SAF).
AMSPORT_R0_NCSVB	511	Port number of the TwinCAT NC Server (Task SVB).
AMSPORT_R0_ISG	550	internal
AMSPORT_R0_CNC	600	Port number of the TwinCAT NC I Server.
AMSPORT_R0_LINE	700	internal
AMSPORT_R0_PLC	800	Port number of the TwinCAT PLC Servers (only on the Buscontroller).
AMSPORT_R0_PLC_RTS1	801	Port number of the TwinCAT 2.xx PLC Server runtime 1
AMSPORT_R0_PLC_RTS2	811	Port number of the TwinCAT 2.xx PLC Server runtime 2
AMSPORT_R0_PLC_RTS3	821	Port number of the TwinCAT 2.xx PLC Server runtime 3
AMSPORT_R0_PLC_RTS4	831	Port number of the TwinCAT 2.xx PLC Server 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	14001	Port number of the TwinCAT Scope Server.

Ads-States

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 (System is in config mode)
ADSSTATE_RECONFIG	16	ADS Status: Reconfiguration (System should restart in config mode)
ADSSTATE_STOPPING	17	
ADSSTATE_INCOMPATIBLE	18	
ADSSTATE_EXCEPTION	19	
ADSSTATE_MAXSTATES	20	Max. number of available ads states

ADS/System Services

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 File Service

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	
SYSTEMSERVICE_FDELETE	131	
SYSTEMSERVICE_FRENAME	132	
SYSTEMSERVICE_REG_HKEYLOCALMACHINE	200	
SYSTEMSERVICE_SENDEMAIL	300	
SYSTEMSERVICE_TIMESERVICES	400	
SYSTEMSERVICE_STARTPROCESS	500	
SYSTEMSERVICE_CHANGENETID	600	

System Servie Timeservices

System Service Index Offsets (Timeservices)	Value	Description
TIMESERVICE_DATEANDTIME	1	
TIMESERVICE_SYSTEMTIMES	2	
TIMESERVICE_RTCTIMEDIFF	3	
TIMESERVICE_ADJUSTTIMETORTC	4	

ADSLOG message types

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	

BOOTDATA flags

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
SYSTEMSTATEFLAGS_RTVIOLATION	16#02	Realtime violation latency overrun

File output modes

Masks for File output	Value	Description
FOPEN_MODEREAD	16#0001	'r': Opens file for reading
FOPEN_MODEWRITE	16#0002	'w': Opens file for reading, (possible) existing files were overwritten.
FOPEN_MODEAPPEND	16#0004	'a': Opens file for reading, is attached to (possible) existing file. If no file existes, it will be created.
FOPEN_MODEPLUS	16#0008	'+': Opens file for reading and writing.
FOPEN_MODEBINARY	16#0010	'b': Opens file forbinary reading and writing.
FOPEN_MODETEXT	16#0020	't': Opens file for textual reading and writing:

Eventlogger constants

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 or 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 formatter.

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

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

6.2 Library version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

Global_Version

```
VAR_GLOBAL CONSTANT
    stLibVersion_Tc2_System : ST_LibVersion;
END_VAR
```

Name	Type	Description
stLibVersion_Tc2_System	ST_LibVersion	Version number of the Tc2_System library (type: ST_LibVersion)

To check whether the version you have is the version you need, use the function F_CmpLibVersion [▶ 86].

All other options for comparing library versions, which you may know from TwinCAT 2, are outdated.

Requirements

Development environment	Target system type	PLC libraries to include (Category group)
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_System (System)

7 Samples

7.1 Example with AdsReadInd /AdsReadRes function blocks

The example shows the implementation of a simple ADS Server application to the PLC. The server application can process the ADSREAD requests of an ADS Client application.

In the example application, ADSREAD requests are used to increment/decrement or reset a PLC counter variable in the PLC task. If successful the value of the counter variable is returned to the ADS Client application

The complete sources of the ADS server application can be unpacked here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/710574987/.zip

An ADS Client application suitable for the ADS Server can be found here: Example with ADSREAD function block. [▶ 135]

ADS Client application

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;

The index offset parameter is zero.

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

PLC program

The ADSREAD requests are intercepted as indications in the PLC task by an instance of the ADSREADIND [▶ 27] function block. Afterwards the index group and index offset parameters and the required data length and validity are checked. In the CASE instruction the desired operation with the PLC variables is carried out. If successful a response is sent back by an instance of the ADSREADRES [▶ 36] function block to the caller with the current value of the PLC variables. In the case of an error an appropriate error message. In the next cycle the CLEAR and RESPOND flags on the function blocks are reset in order to be able to process further indications.

Declaration Part

```
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        : PVOID;(* Pointer to returned read data/buffer *)
    nErrID       : UDINT;(* Read indication result error code *)
    nCounter     : INT;(* Server data *)
END_VAR
```

Implementation

```
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     := DEVICE_SRVNOTSUPP;


     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 := NOERR;
                         ELSE (* ADS error (example): Invalid size *)
                              nErrID := DEVICE_INVALIDSIZE;
                         END_IF
               ELSE (* ADS error (example): Invalid index offset *)
                    nErrID := DEVICE_INVALIDOFFSET;
               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 := NOERR;
                         ELSE(* ADS error (example): Invalid size *)
                              nErrID := DEVICE_INVALIDSIZE;
                         END_IF
               ELSE (* ADS error (example): Invalid index offset *)
                    nErrID := DEVICE_INVALIDOFFSET;
               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 := NOERR;
                         ELSE(* ADS error (example): Invalid size *)
                              nErrID := DEVICE_INVALIDSIZE;
                         END_IF
               ELSE (* ADS error (example): ervice is not supported by server *)
                    nErrID := DEVICE_SRVNOTSUPP;
               END_CASE

     ELSE (* ADS error (example): Invalid index group *)
          nErrID := DEVICE_INVALIDGRP;
     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
```

7.2 Example with AdsWriteInd/AdsWriteRes function blocks

The example shows the implementation of a simple ADS Server application to the PLC. The server application can process the ADSWRITE requests of an ADS Client application.

In the example application ADSWRITE requests are used to transfer arrays with integer values to the PLC task. The received data are copied in the PLC into an appropriate array variable.

The complete sources of the ADS server application can be unpacked here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/710582923/.zip

An ADS Client application suitable for the ADS Server can be found here: Example with ADSWRITE function block. [▶ 136]

ADS Client application

The desired service/command 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 to 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.

PLC program

The requests are intercepted as indications in the PLC task by an instance of the ADSWRITEIND [▶ 30] 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 [▶ 37]-function block. In the next cycle 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 (== ZERO).

For this reason the sent data is first copied into the PLC variable before the CLEAR input is set to TRUE.

Declaration Part

```
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      : PVOID;(* Pointer to written data buffer *)
   nResult     : UDINT;(* Write indication result error code *)
   arrInt      : ARRAY[0..9] OF INT;(* Server data *)
END_VAR
```

Implementation

```
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      := DEVICE_SRVNOTSUPP;

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

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

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

7.3 Example with AdsRead function block

The example demonstrates the use of the ADSREAD function block in an ADS Client application.

The complete sources of the ADS Client application can be unpacked here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/710578827/.zip

Declaration Part

```
PROGRAM MAIN
VAR
    fbReadReq  : ADSREADEX := ( NETID := '', PORT := 851, TMOUT := DEFAULT_ADS_TIMEOUT );
    bIncrement : BOOL;(* Rising edge at this variable starts command execution *)
    bDecrement : BOOL;(* Rising edge at this variable starts command execution *)
    bReset     : BOOL;(* Rising edge at this variable starts command execution *)
    bOther     : BOOL;(* Rising edge at this variable starts command execution *)

    nState     : BYTE;
    bBusy      : BOOL;
    bError     : BOOL;
    nErrID     : UDINT;
    cbRead     : UDINT;

    nCounter   : INT;(* Server data to be read *)
END_VAR
```

Implementation

```
CASE nState OF
    0:
        IF bIncrement OR bDecrement OR bReset OR bOther THEN
            bBusy := TRUE;
            bError := FALSE;
            nErrID := 0;

            fbReadReq( READ := FALSE );

            IF bIncrement THEN(* Incement counter value *)
                bIncrement := FALSE;
                fbReadReq( IDXGRP := 16#80000001, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            ELSIF bDecrement THEN(* Decrement counter value *)
                bDecrement := FALSE;
                fbReadReq( IDXGRP := 16#80000002, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            ELSIF bReset THEN(* Reset counter value *)
```

```
                bReset := FALSE;
                fbReadReq( IDXGRP := 16#80000003, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            ELSIF bOther THEN(* Call unsupported function *)
                bOther := FALSE;
                fbReadReq( IDXGRP := 16#80000004, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            END_IF

            nState := 1;
        END_IF
    1:
        fbReadReq( READ := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID, COUNT_R=>cbRead );
        IF NOT bBusy THEN
            IF NOT bError THEN
                nState := 0;(* Success *)
            ELSE
                nState := 100;(* Error *)
            END_IF
        END_IF

    100:(* TODO::Implement error handler *)
        nState := 0;

END_CASE
```

7.4 Example with AdsWrite function block

The example demonstrates the use of the ADSWRITE function block in an ADS Client application.

The complete sources of the ADS Client application can be unpacked here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/710586763/.zip

Declaration Part

```
PROGRAM MAIN
VAR
   fbWriteReq   : ADSWRITE := ( NETID := '', PORT := 851, TMOUT := DEFAULT_ADS_TIMEOUT );
   bWrite       : BOOL;(* Rising edge at this variable starts command execution *)
   nState       : BYTE;
   bBusy        : BOOL;
   bError       : BOOL;
   nErrID       : UDINT;
   arrInt       : ARRAY[0..9] OF INT;(* Server data to be written *)
   i            : INT;
END_VAR
```

Implementation

```
FOR i:=0 TO 9 BY 1 DO (* modify/simulate new data *)
   arrInt[i] := arrInt[i] + 1;
END_FOR


CASE nState OF
   0:
      IF bWrite THEN
         bWrite := FALSE;

         bBusy := TRUE;
         bError := FALSE;
         nErrID := 0;

         fbWriteReq( WRITE := FALSE );
         fbWriteReq( IDXGRP := 16#80000005, IDXOFFS := 7,
                  LEN := SIZEOF( arrInt ), SRCADDR := ADR( arrInt ),
                  WRITE := TRUE );
         nState := 1;
      END_IF

   1:
      fbWriteReq( WRITE := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID );
      IF NOT bBusy THEN
         IF NOT bError THEN
            nState := 0;(* Success *)
         ELSE
            nState := 100;(* Error *)
```

```
        END_IF
      END_IF

   100:(* TODO::Implement error handler *)
      nState := 0;

END_CASE
```

7.5 Sending/acknowledging EventLogger signals from the PLC

The example demonstrates the use of the ADSLOGEVENT function block.

The complete sources for the example application can be unpacked here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/711421451/.zip

Step by step sequence

Configuration of an event:

Parameterisation of EventConfigData [▶ 124] structure

Transfer of parameters:

Generate an address of a structure, an array or a single variable with ADR operator at EventDataAddress. Determine the length of the structure, array or single variable using the SIZEOF operator and apply it to the EventDataLength input. If, for example, a structure with an INT and an LREAL variable is to be transferred with the event, then a structure must be created with these two components and instanced. The address and the length of this instance must be transferred.

Setting an event:

Rising edge at the Event input

Resetting an event:

Falling edge at the Event input

To acknowledge an events:

Rising edge at the Quit input

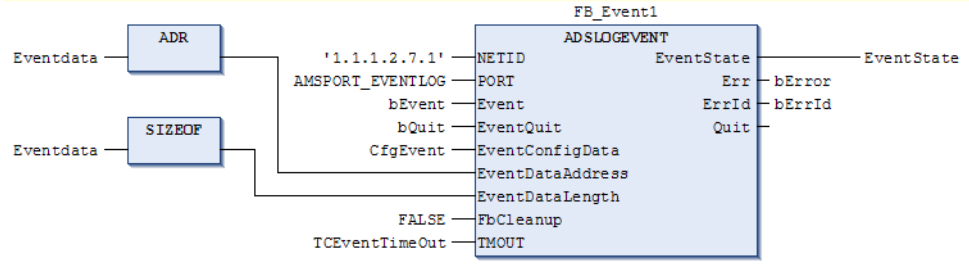
Complete deletion of the instance:

The contents of the instance are completely deleted with a rising edge at the FbCleanup input. An existing event from the EventLogger is not directly deleted by this.

After an event has been sent to the EventLogger, the status of the event [▶ 126] changes visibly at the Eventstate output.

Calling the ADSLOGEVENT function block

```
PROGRAM MAIN
VAR
    FB_Event1   : ADSLOGEVENT;
    CfgEvent    : TcEvent;
    Eventdata   : ParaStruct;
    EventState  : UDINT;
    bEvent      : BOOL;
    bQuit       : BOOL;
END_VAR
VAR CONSTANT
    TCEventDataFormatString : STRING:='%f%d';
    TcEventTimeOut     : TIME:=T#1s;
END_VAR
```



Declaration Part

```
PROGRAM MAIN
VAR
    CfgEvent                : TcEvent;
    fbEvent                 : ADSLOGEVENT;
    bSetEvent               : BOOL;(* Rising edge sets event *)

    eventData               : ST_EventData;
    TcEventDataFormatString : STRING    := '%f%d';
END_VAR
```

Implementation

```
CfgEvent.Class := TCEVENTCLASS_ALARM;
CfgEvent.Prio := 2;
CfgEvent.Id := 1;
CfgEvent.SourceId := 100;
CfgEvent.bQuitRequired := TRUE;
CfgEvent.DataFormatStrAddress := ADR(TcEventDataFormatString);
CfgEvent.Flags := TCEVENTFLAG_LOG OR TCEVENTFLAG_SRCID OR TCEVENTFLAG_AUTOFMTALL;
CfgEvent.StreamType := TCEVENTSTREAM_SIMPLE;
CfgEvent.ProgId :='TcEventFormatter.TcXMLFormatter' ;

eventData.rVal     := 2.65;
eventData.iVal    := 3;

fbEvent( NETID:= '',
        PORT:= 110,
        Event:= bSetEvent,
        EventConfigData:= CfgEvent,
        EventDataAddress := ADR(eventData) ,
        EventDataLength := SIZEOF(eventData),
        TMOUT:= t#3s);
```

7.6 File access from the PLC

The use of the PLC function blocks for data access from the Tc2_system library is introduced in this example. A new function block, FB_FileCopy, is implemented with the aid of the existing function blocks. Using the FB_FileCopy function block, binary files can be copied in the local TwinCAT system or between a local and a remote TwinCAT system.

The complete source code for the example project can be unpacked from here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/707895179/.zip

Network drives cannot be accessed using the FB_FileCopy function block.

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

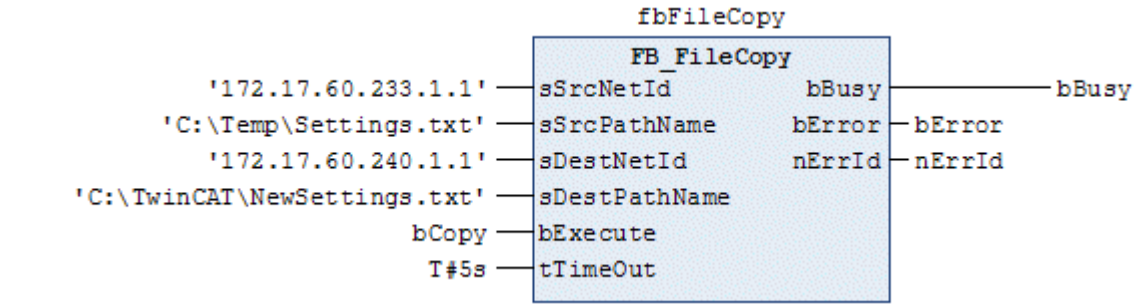
d) Check whether the end of the source file has been reached. If not, then repeat b) and c). If yes, then jump to e)

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.

PLC program

```
PROGRAM MAIN
VAR
    fbFileCopy  : FB_FileCopy;
    bCopy       : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```



Declaration Part

```
FUNCTION_BLOCK FB_FileCopy
VAR_INPUT
    sSrcNetId           : T_AmsNetId;
    sSrcPathName        : T_MaxString;
    sDestNetId          : T_AmsNetId;
    sDestPathName       : T_MaxString;
    bExecute            : BOOL;
    tTimeOut            : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrId              : UDINT;
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
```

7.7 Testing the CPU reserve of a CX70xx

This example is a test project for testing the CPU reserve of a CX70xx. The included function block `FB_Test_CPU_Performance` measures the CPU reserve you have with your application. The function block reads the current CPU power and cycle time. The function block then increases the CPU load until the CX7k no longer operates in real time. Then it reduces the load again until a stable real time is reached. The function block then determines the CPU power and the cycle time and offsets them against the time and load taken at the start of the measurement and gives you the delta. Use the function block for test purposes only and not in a real environment.

If the CPU reserve is greater than 20%, you can make the task cycle time faster than the one currently in use. The advantages of a faster task are faster reaction to inputs and, depending on the program content, a faster application. A few milliseconds can add up to increase the output of a machine. A reserve power of 20 % is ideal.

The sources for the sample project can be unpacked here: https://infosys.beckhoff.com/content/1033/TcPlcLib_Tc2_System/Resources/11298888331/.zip

Falsified measurement result

If the function block is used in a low-priority task, the result is falsified and the real data cannot be determined. Due to the long measurement, the slow tasks are also taken into account.

- If possible, always use the function block in the fastest high-priority task.

Loops or program sections not run through during the measurement falsify the measurement

Loops, multiple tasks and thus strongly fluctuating cycle times cause a strongly fluctuating CPU load.

- Set the TimeOut of the function block to a larger value, because the function block searches for the highest CPU load and then takes longer than with a constant CPU load.

Aborting the measurement

The function block can only be used if there are no real-time violations in your configuration. If the function block reads out real-time violations already at start-up, the function block aborts the measurement.

Information on the function block `FB_Test_CPU_Performance`:

VAR_INPUT

Name	Type	Description
bExecute	BOOL	A positive edge activates the function block.
tTimeOut	TIME	Time to stop the measurement if exceeded.

VAR_OUTPUT

Name	Type	Description
bBusy	BOOL	The function block is active and working.
bError	BOOL	The function block has an error.
nErrorID	UDINT	ADS error code
nCpuLoadReserve	UDINT	Reserve of CPU in [%]
fCycleTimeReserve	LREAL	Cycle time reserve in [ms]

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.22	PC or CX (x86, x64, ARM)	Tc2_System (system) >= 3.4.25.0

8 Appendix

8.1 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 143]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 143]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 144]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 146]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was cancelled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

More Information:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com