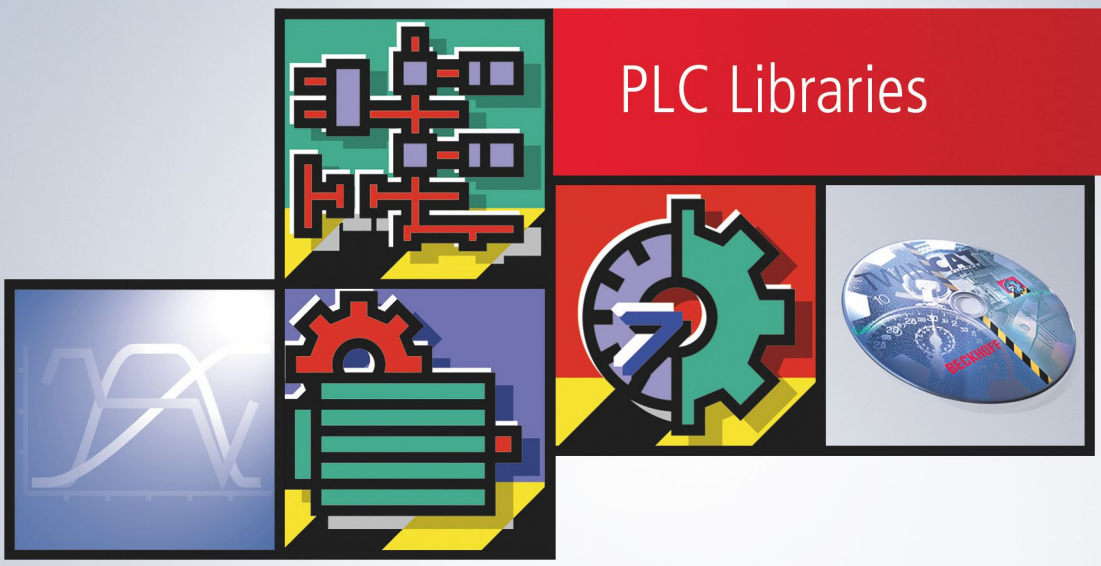
BECKHOFF New Automation Technology

Manual | EN

TX1200

TwinCAT 2 | PLC Library: TcUtilities



2025-02-26 | Version: 1.4

Table of contents

1 Foreword 11
1.1 Notes on the documentation 11
1.2 For your safety 12
1.3 Notes on information security 13

2 Overview 14

3 Functions 20
3.1 DT_TO_SYSTEMTIME 20
3.2 DT_TO_FILETIME 20
3.3 SYSTEMTIME_TO_DT 21
3.4 SYSTEMTIME_TO_FILETIME 22
3.5 SYSTEMTIME_TO_STRING 22
3.6 STRING_TO_SYSTEMTIME 23
3.7 FILETIME_TO_DT 24
3.8 FILETIME_TO_SYSTEMTIME 24
3.9 TIME_TO_OTSTRUCT 25
3.10 OTSTRUCT_TO_TIME 26
3.11 DEG_TO_RAD 26
3.12 RAD_TO_DEG 27
3.13 BYTE_TO_DECSTR 27
3.14 BYTE_TO_BINSTR 28
3.15 BYTE_TO_HEXSTR 28
3.16 BYTE_TO_OCTSTR 29
3.17 WORD_TO_DECSTR 29
3.18 WORD_TO_BINSTR 30
3.19 WORD_TO_HEXSTR 30
3.20 WORD_TO_OCTSTR 31
3.21 PVOID_TO_DECSTR 32
3.22 PVOID_TO_BINSTR 33
3.23 PVOID_TO_HEXSTR 34
3.24 PVOID_TO_OCTSTR 35
3.25 DWORD_TO_DECSTR 36
3.26 DWORD_TO_BINSTR 37
3.27 DWORD_TO_HEXSTR 38
3.28 DWORD_TO_OCTSTR 39
3.29 STRING_TO_PVOID 40
3.30 PVOID_TO_STRING 41
3.31 STRING_TO_GUID 42
3.32 GUID_TO_STRING 42
3.33 REGSTRING_TO_GUID 43
3.34 GUID_TO_REGSTRING 43
3.35 GuidsEqualByVal 44
3.36 HEXASCNIBBLE_TO_BYTE 44
3.37 HEXCHRNIBBLE_TO_BYTE 45
3.38 DINT_TO_DECSTR 45

3.39 LREAL_TO_FMTSTR ... 46
3.40 ROUTETRANSPORT_TO_STRING ... 48
3.41 BYTEARR_TO_MAXSTRING ... 48
3.42 MAXSTRING_TO_BYTEARR ... 49
3.43 DATA_TO_HEXSTR ... 49
3.44 HEXSTR_TO_DATA ... 50
3.45 IsFinite ... 51
3.46 F_CheckSum16 ... 53
3.47 F_DATA_TO_CRC16_CCITT ... 53
3.48 F_BYTE_TO_CRC16_CCITT ... 54
3.49 F_SwapRealEx ... 54
3.50 F_FormatArgToStr ... 55
3.51 F_GetMaxMonthDays ... 58
3.52 F_GetDOYOfYearMonthDay ... 59
3.53 F_GetMonthOfDOY ... 59
3.54 F_YearIsLeapYear ... 60
3.55 F_GetDayOfMonthEx ... 61
3.56 F_TranslateFileTimeBias ... 62
3.57 F_ToUCase ... 63
3.58 F_ToLCase ... 64
3.59 F_LTrim ... 64
3.60 F_RTrim ... 65
3.61 F_GetDayOfWeek ... 66
3.62 F_GetWeekOfTheYear ... 66
3.63 F_CreateHashTableHnd ... 67
3.64 F_CreateLinkedListHnd ... 68
3.65 CSVFIELD_TO_STRING ... 69
3.66 CSVFIELD_TO_ARG ... 70
3.67 STRING_TO_CSVFIELD ... 71
3.68 ARG_TO_CSVFIELD ... 72
3.69 F_GetVersionTcUtilities ... 74
3.70 BYTE_TO_LREALEX ... 74
3.71 DWORD_TO_LREALEX ... 75
3.72 UDINT_TO_LREALEX ... 76
3.73 UINT_TO_LREALEX ... 77
3.74 USINT_TO_LREALEX ... 78
3.75 WORD_TO_LREALEX ... 79
3.76 Unsigned 64 bit integer ... 80
3.76.1 ULARGE_INTEGER ... 80
3.76.2 UInt64Add64 ... 80
3.76.3 UInt64Add64Ex ... 81
3.76.4 UInt64Sub64 ... 81
3.76.5 UInt32x32To64 ... 82
3.76.6 UInt64Mul64 ... 82
3.76.7 UInt64Mul64Ex ... 83
3.76.8 UInt64Div64 ... 83

3.76.9 UInt64Div64Ex ... 84
3.76.10 UInt64Div16Ex ... 84
3.76.11 UInt64Mod64 ... 85
3.76.12 UInt64Cmp64 ... 85
3.76.13 UInt64And ... 86
3.76.14 UInt64Or ... 86
3.76.15 UInt64Xor ... 87
3.76.16 UInt64Not ... 87
3.76.17 UInt64Min ... 87
3.76.18 UInt64Max ... 88
3.76.19 UInt64Limit ... 88
3.76.20 UInt64Shl ... 89
3.76.21 UInt64Shr ... 89
3.76.22 UInt64Rol ... 90
3.76.23 UInt64Ror ... 90
3.76.24 UInt64isZero ... 91
3.76.25 LREAL_TO_UINT64 ... 91
3.76.26 STRING_TO_UINT64 ... 91
3.76.27 UINT64_TO_LREAL ... 92
3.76.28 UINT64_TO_STRING ... 92
3.77 Signed 64 bit integer ... 93
3.77.1 LARGE_INTEGER ... 93
3.77.2 Int64Add64 ... 93
3.77.3 Int64Add64Ex ... 94
3.77.4 Int64Sub64 ... 94
3.77.5 Int64Div64Ex ... 94
3.77.6 Int64Cmp64 ... 95
3.77.7 Int64Not ... 95
3.77.8 Int64Negate ... 96
3.77.9 Int64isZero ... 96
3.77.10 LREAL_TO_INT64 ... 96
3.77.11 INT64_TO_LREAL ... 97
3.77.12 ULARGE_TO_LARGE ... 97
3.77.13 LARGE_TO_ULARGE ... 98
3.78 Signed 16 bit fixed point ... 98
3.78.1 LREAL_TO_FIX16 ... 98
3.78.2 FIX16_TO_LREAL ... 99
3.78.3 WORD_TO_FIX16 ... 99
3.78.4 FIX16_TO_WORD ... 100
3.78.5 FIX16Add ... 100
3.78.6 FIX16Align ... 101
3.78.7 FIX16Div ... 102
3.78.8 FIX16Mul ... 103
3.78.9 FIX16Sub ... 103
3.79 Helper functions ... 104
3.79.1 F_LREAL ... 104

3.79.2 F_REAL ... 104
3.79.3 F_BYTE ... 105
3.79.4 F_WORD ... 105
3.79.5 F_DWORD ... 106
3.79.6 F_SINT ... 106
3.79.7 F_INT ... 107
3.79.8 F_DINT ... 107
3.79.9 F_USINT ... 108
3.79.10 F_UINT ... 108
3.79.11 F_UDINT ... 109
3.79.12 F_STRING ... 109
3.79.13 F_BOOL ... 109
3.79.14 F_BIGTYPE ... 110
3.79.15 F_LARGE ... 110
3.79.16 F_HUGE ... 111
3.79.17 F_ULARGE ... 111
3.79.18 F_UHUGE ... 112
3.79.19 F_PVOID ... 112
3.79.20 F_ARGCPY ... 113
3.79.21 F_ARGCMP ... 113
3.79.22 F_ARGISZERO ... 114
3.80 P[TYPE]_TO_[TYPE] converting functions ... 114
3.80.1 PBOOL_TO_BOOL ... 114
3.80.2 PBYTE_TO_BYTE ... 115
3.80.3 PDATE_TO_DATE ... 115
3.80.4 PDINT_TO_DINT ... 115
3.80.5 PDT_TO_DT ... 116
3.80.6 PDWORD_TO_DWORD ... 116
3.80.7 PINT_TO_INT ... 116
3.80.8 PLREAL_TO_LREAL ... 117
3.80.9 PMAXSTRING_TO_MAXSTRING ... 117
3.80.10 PREAL_TO_REAL ... 117
3.80.11 PSINT_TO_SINT ... 118
3.80.12 PSTRING_TO_STRING ... 118
3.80.13 PTIME_TO_TIME ... 118
3.80.14 PTOD_TO_TOD ... 119
3.80.15 PUDINT_TO_UDINT ... 119
3.80.16 PUINT_TO_UINT ... 119
3.80.17 PUSINT_TO_USINT ... 120
3.80.18 PWORD_TO_WORD ... 120
3.80.19 PLARGE_TO_LARGE ... 120
3.80.20 PHUGE_TO_HUGE ... 121
3.80.21 PULARGE_TO_ULARGE ... 121
3.80.22 PUHUGE_TO_UHUGE ... 122
3.81 Byte order converting functions ... 122
3.81.1 Host Byte Order / Network Byte Order ... 122

3.81.2 HOST_TO_BE16 ... 122
3.81.3 HOST_TO_BE32 ... 123
3.81.4 HOST_TO_BE64 ... 123
3.81.5 HOST_TO_BE128 ... 124
3.81.6 BE16_TO_HOST ... 124
3.81.7 BE32_TO_HOST ... 124
3.81.8 BE64_TO_HOST ... 125
3.81.9 BE128_TO_HOST ... 125

4 Function blocks ... 126
4.1 NT_Shutdown ... 126
4.2 NT_AbortShutdown ... 127
4.3 NT_Reboot ... 128
4.4 NT_GetTime ... 129
4.5 NT_SetLocalTime ... 130
4.6 NT_StartProcess ... 131
4.7 NT_SetTimeToRTCTime ... 133
4.8 PLC_Reset ... 134
4.9 PLC_Start ... 135
4.10 PLC_Stop ... 136
4.11 PLC_ReadSymInfo ... 137
4.12 PLC_ReadSymInfoByName ... 138
4.13 PLC_ReadSymInfoByNameEx ... 140
4.14 FB_RegQueryValue ... 141
4.15 FB_RegSetValue ... 144
4.16 FB_GetRouterStatusInfo ... 146
4.17 FB_EnumRouteEntry ... 147
4.18 FB_AddRouteEntry ... 149
4.19 FB_RemoveRouteEntry ... 150
4.20 FB_WritePersistentData ... 152
4.21 FB_BasicPID ... 153
4.22 FB_GetLocalAmsNetId ... 154
4.23 FB_AmsLogger ... 155
4.24 FB_FormatString ... 157
4.25 FB_EnumFindFileEntry ... 158
4.26 FB_EnumFindFileList ... 160
4.27 FB_EnumStringNumbers ... 161
4.28 FB_GetHostName ... 163
4.29 FB_GetHostAddrByName ... 164
4.30 FB_GetAdaptersInfo ... 165
4.31 FB_FileRingBuffer ... 166
4.32 FB_MemRingBuffer ... 169
4.33 FB_MemRingBufferEx ... 170
4.34 FB_MemBufferSplit ... 171
4.35 FB_MemBufferMerge ... 173
4.36 FB_MemStackBuffer ... 174
4.37 FB_StringRingBuffer ... 176

4.38 FB_GetTimeZoneInformation ... 177
4.39 FB_SetTimeZoneInformation ... 178
4.40 FB_LocalSystemTime ... 180
4.41 FB_TzSpecificLocalTimeToSystemTime ... 183
4.42 FB_TzSpecificLocalTimeToFileTime ... 184
4.43 FB_FileTimeToTzSpecificLocalTime ... 187
4.44 FB_SystemTimeToTzSpecificLocalTime ... 189
4.45 FB_HashTableCtrl ... 190
4.46 FB_LinkedListCtrl ... 192
4.47 RestartTwinCAT ... 193
4.48 TC_Stop ... 194
4.49 TC_Config ... 195
4.50 TC_CpuUsage ... 196
4.51 TC_SysLatency ... 197
4.52 ScopeLoadFile ... 198
4.53 ScopeSetOnline ... 199
4.54 ScopeSetOffline ... 200
4.55 ScopeGetState ... 200
4.56 ScopeManualTrigger ... 201
4.57 ScopeSetRecordLen ... 202
4.58 ScopeGetRecordLen ... 203
4.59 ScopeASCIIExport ... 204
4.60 ScopeViewExport ... 204
4.61 ScopeSaveAs ... 205
4.62 ScopeExit ... 206
4.63 FB_ScopeServerControl ... 207
4.64 DEC_TO_BCD ... 210
4.65 BCD_TO_DEC ... 211
4.66 RTC ... 212
4.67 RTC_EX ... 213
4.68 RTC_EX2 ... 214
4.69 GetRemotePCInfo ... 215
4.70 WritePersistentData ... 216
4.71 Profiler ... 218
4.72 DCF77_TIME ... 221
4.73 DCF77_TIME_EX ... 224
4.74 FB_CSVMemBufferReader ... 227
4.75 FB_CSVMemBufferWriter ... 228

5 Data types ... 230
5.1 TIMESTRUCT ... 230
5.2 OTSTRUCT ... 230
5.3 PROFILERSTRUCT ... 231
5.4 REMOTEPC ... 231
5.5 REMOTEPCINFOSTRUCT ... 232
5.6 SYMINFOSTRUCT ... 232
5.7 ADSDATATYPEID ... 233

5.8 E_RegValueType ... 233
5.9 E_ArgType ... 234
5.10 E_AmsLoggerMode ... 234
5.11 E_PersistentMode ... 235
5.12 E_TypeFieldParam ... 235
5.13 E_EnumCmdType ... 235
5.14 E_RouteTransportType ... 236
5.15 E_NumGroupTypes ... 236
5.16 E_MIB_IF_Type ... 237
5.17 E_TimeZoneID ... 237
5.18 E_SBCSType ... 237
5.19 E_DbgContext ... 238
5.20 E_DbgDirection ... 238
5.21 T_Arg ... 239
5.22 T_FILETIME ... 239
5.23 T_ULARGE_INTEGER ... 239
5.24 T_UHUGE_INTEGER ... 240
5.25 T_LARGE_INTEGER ... 240
5.26 T_HUGE_INTEGER ... 240
5.27 T_FIX16 ... 240
5.28 T_HHASHTABLE ... 242
5.29 T_HLINKEDLIST ... 242
5.30 T_HashTableEntry ... 242
5.31 T_LinkedListEntry ... 243
5.32 ST_TcRouterStatusInfo ... 243
5.33 ST_AmsRouteEntry ... 243
5.34 ST_FindFileEntry ... 244
5.35 ST_FileAttributes ... 244
5.36 ST_IPAdapterInfo ... 245
5.37 ST_IPAdapterHwAddr ... 246
5.38 ST_FileRBufferHead ... 247
5.39 ST_TimeZoneInformation ... 247
5.40 GUID ... 248

6 Global variables/constants ... 250
6.1 Global Variables ... 250
6.2 Format error codes ... 250
6.3 Error codes TcUtilities ... 251

7 Samples ... 252
7.1 Example: Communication BC/BX<->PC/CX (F_SwapRealEx) ... 252
7.2 Example: File search (FB_EnumFindFileEntry, FB_EnumFindFileList) ... 255
7.3 Example: File ring FiFo (FB_FileRingBuffer) ... 257
7.4 Example: Memory ring FiFo (FB_MemRingBuffer) ... 258
7.5 Example: Memory ring FiFo (FB_MemRingBufferEx) ... 260
7.6 Example: Hash table (FB_HashTableCtrl) ... 261
7.7 Example: Linked list (FB_LinkedListCtrl) ... 265

7.8 Example: Writing/reading of CSV file 270
7.9 Example: Software clocks (RTC, RTC_EX, RTC_EX2) 273

8 Appendix 275
8.1 Format specification 275
8.2 Writing of persistent data: System behaviour 277

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.
The documentation and the following notes and explanations must be complied with when installing and commissioning the components.
The trained specialists must always use the current valid documentation.

The trained specialists must ensure that the application and use of the products described is in line with all safety requirements, including all relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been compiled 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.
Claims to modify products that have already been supplied may not 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 the designations or trademarks contained in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The distribution and reproduction of this document, as well as the use and communication of its contents without express authorization, are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

Third-party trademarks

Trademarks of third parties may be used in this documentation. You can find the trademark notices here: https://www.beckhoff.com/trademarks.

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 in our https://www.beckhoff.com/secguide.

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

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

2 Overview

The utilities library contains a number of useful function blocks and functions with which TwinCAT PLC or operating system functions can be called. The function blocks are based, internally, on the system library.

By adding **TcUtilities.Lib** the following libraries are included automatically: Standard.Lib; TcBase.Lib; TcSystem.Lib;

Operating system functions

Name	Description
NT_Shutdown [▶ 126]	Shuts down the operating system
NT_AbortShutdown [▶ 127]	Aborts operating system shutdown command
NT_Reboot [▶ 128]	Restarts the operating system
NT_GetTime [▶ 129]	Reads the local Windows system time
NT_SetLocalTime [▶ 130]	Sets the local Windows system time
NT_StartProcess [▶ 131]	Starts a Windows application from the PLC
NT_SetTimeToRTCTime [▶ 133]	Synchronize the local Windows system time to the real-time clock of the PC
FB_RegQueryValue [▶ 141]	Reads system registry
FB_RegSetValue [▶ 144]	Writes system registry
FB_EnumFindFileEntry [▶ 158]	This function block searches a directory for a file or subdirectory whose name matches the specified file name.
FB_EnumFindFileList [▶ 160]	This function block searches a directory for a file or subdirectory whose name matches the specified file name. Found entries are read block by block.
FB_GetAdaptersInfo [▶ 165]	The function block retrieves adapter information for the local or remote computer.
FB_GetHostName [▶ 163]	The function block returns the standard host name for the local or remote machine.
FB_GetHostAddrByName [▶ 164]	Converts the host name to the (IPv4) Internet Protocol network address.
FB_GetTimeZoneInformation [▶ 177]	Reads operating system time zone settings.
FB_SetTimeZoneInformation [▶ 178]	Writes operating system time zone settings.
FB_LocalSystemTime [▶ 180]	Reads local Windows System Time and Daylight-/Standard-time information.

TwinCAT PLC functions

Name	Description
PLC_Reset [▶ 134]	Resets the PLC
PLC_Start [▶ 135]	Starts the PLC
PLC_Stop [▶ 136]	Stops the PLC
PLC_ReadSymInfo [▶ 137]	Reads the PLC symbol information
PLC_ReadSymInfoByName [▶ 138]	Reads the additional symbol information knowing the symbol name
PLC_ReadSymInfoByNameEx [▶ 140]	Reads the additional symbol information knowing the symbol name and cuts comment string if the string exceeds the available buffer length.
Profiler [▶ 218]	Reads the execution time of PLC program
WritePersistentData [▶ 216]	Writes persistent data

Name	Description
FB_WritePersistentData [▶ 152]	Writes persistent data (Extendedversion)

Checksum/CRC functions

Name	Description
F_CheckSum16 [▶ 53]	Calculates the 16 Bit CheckSum
F_DATA_TO_CRC16_CCITT [▶ 53]	Calculates the CRC16-CCITT (cyclic redundancy check) of any data type
F_BYTE_TO_CRC16_CCITT [▶ 54]	Calculates the CRC16-CCITT (cyclic redundancy check) of single data byte

TwinCAT system functions

Name	Description
TC_Restart [▶ 193]	Restarts the TwinCAT system
TC_Stop [▶ 194]	Stops the TwinCAT system
TC_Config [▶ 195]	Starts the TwinCAT system in Config mode.
TC_CpuUsage [▶ 194]	Reads the CPU usage of the TwinCAT system
TC_SysLatency [▶ 197]	Reads the TwinCAT System latency time
GetRemotePCInfo [▶ 215]	Reads the TwinCAT Router information about configured remote PCs
FB_GetLocalAmsNetId [▶ 154]	Reads the local AmsNetId (TwinCAT PC network address)
FB_GetRouterStatusInfo [▶ 146]	Reads TwinCAT Router status information
FB_EnumRouteEntry [▶ 147]	Enumerates TwinCAT Router connections
FB_AddRouteEntry [▶ 149]	Adds one TwinCAT Router connection
FB_RemoveRouteEntry [▶ 150]	Removes one TwinCAT Router connection

TwinCAT Scope View functions

Some functions can be used if only one View in the TwinCAT Scope View application is activated.

Name	Description
ScopeLoadFile [▶ 198]	Loads Scope View Configuration Project (*.scp)
ScopeSetOnline [▶ 199]	Switches Scope View in Online state
ScopeSetOffline [▶ 200]	Switches Scope View in Offline state
ScopeGetState [▶ 200]	Reads Scope View current state
ScopeManualTrigger [▶ 201]	Performs manual trigger
ScopeSetRecordLen [▶ 202]	Sets the record time
ScopeGetRecordLen [▶ 203]	Reads the record time
ScopeASCIIExport [▶ 204]	Exports Scope View in ASCII file
ScopeViewExport [▶ 204]	Exports Scope View in binary file
ScopeSaveAs [▶ 205]	Saves Scope View Project file
ScopeExit [▶ 206]	Terminates Scope View instance (exe)

TwinCAT Scope Server

since library version 2.0.52

Name	Description
FB_ScopeServerControl	Controls (start/save..) the Scope Server for data logging

TwinCAT ADS Monitor functions

Name	Description
FB_AmsLogger [▶ 155]	Start or stop the TwinCAT AMS logger

Converting functions

Name	Description
DT_TO_SYSTEMTIME [▶ 20]	Converts DATE_AND_TIME to the Windows system time structure
DT_TO_FILETIME [▶ 20]	Converts DATE_AND_TIME to file time
SYSTEMTIME_TO_DT [▶ 21]	Converts Windows system time structure to DATE_AND_TIME
SYSTEMTIME_TO_FILETIME [▶ 22]	Converts Windows system time structure to file time.
SYSTEMTIME_TO_STRING [▶ 22]	Converts Windows system time structure to string.
STRING_TO_SYSTEMTIME [▶ 23]	Converts string to Windows system time structure.
FILETIME_TO_DT [▶ 24]	Converts Windows file time to DATE_AND_TIME.
FILETIME_TO_SYSTEMTIME [▶ 24]	Converts Windows file time to Windows system time structure.
DEC_TO_BCD [▶ 210]	Converts decimal numbers to BCD numbers
BCD_TO_DEC [▶ 211]	Converts BCD numbers to decimal numbers
DEG_TO_RAD [▶ 26]	Converts degrees to radiants
RAD_TO_DEG [▶ 27]	Converts radiants to degrees
TIME_TO_OTSTRUCT [▶ 25]	Converts TIME variable to structure with milliseconds, seconds, minutes, hours, days and weeks
OTSTRUCT_TO_TIME [▶ 26]	Converts structure with milliseconds, seconds, minutes, hours, days and weeks to TIME variable
F_SwapRealEx [▶ 54]	Converts bus controller REAL number to Intel PC REAL number format
BYTEARR_TO_MAXSTRING [▶ 48]	Converts byte array to string
MAXSTRING_TO_BYTEARR [▶ 49]	Converts string to byte array
F_TranslateFileTimeBias [▶ 62]	Converts UTC time to local time and vice versa (using bias value)
FB_TzSpecificLocalTimeToFileTime [▶ 184]	Converts local time (file time format) to UTC time
FB_TzSpecificLocalTimeToSystemTime [▶ 183]	Converts local time (structured system time format) to UTC time
FB_FileTimeToTzSpecificLocalTime [▶ 187]	Converts UTC time (file time format) to local time
FB_SystemTimeToTzSpecificLocalTime [▶ 189]	Converts UTC time (structured system time format) to local time

String format functions

Name	Description
LREAL_TO_FMTSTR [▶ 46]	Converts floating point numbers to string
DWORD_TO_DECSTR [▶ 36]	Converts decimal numbers to decimal string
DWORD_TO_HEXSTR [▶ 38]	Converts decimal numbers to hexadecimal string
DWORD_TO_OCTSTR [▶ 39]	Converts decimal number to octal string
DWORD_TO_BINSTR [▶ 37]	Converts decimal number to binary string

Name	Description
DINT_TO_DECSTR [▶ 45]	Converts signed decimal number to decimal string
F_FormatArgToStr [▶ 55]	Converts one argument (plc variable) to string
FB_FormatString [▶ 157]	Converts (formats) up to 10 arguments (plc variables) to string
FB_EnumStringNumbers [▶ 161]	Search for numbers in string
F_ToUCase [▶ 63]	Converts a specified string to uppercase
F_ToLCase [▶ 64]	Converts a specified string to lowercase
F_LTrim [▶ 64]	Removes spaces on the left side of a string
F_RTrim [▶ 65]	Removes spaces on the right side of a string
DATA_TO_HEXSTR [▶ 49]	Converts binary data to hex string

64 bit functions (unsigned)

Name	Description
ULARGE_INTEGER [▶ 80]	Initializes/sets 64 bit number value
UInt64Add64 [▶ 80]	Adds two 64 bit numbers
UInt64Add64Ex [▶ 81]	Adds two 64 bit numbers (with overflow check)
UInt64Sub64 [▶ 81]	Subtracts two 64 bit numbers
UInt64Cmp64 [▶ 85]	Compares two 64 bit numbers
UInt32x32To64 [▶ 82]	Multiplies two 32 bit numbers. Result is 64 bit number.
UInt64Mul64 [▶ 82]	Multiplies two 64 bit numbers. Result is 64 bit number.
UInt64Mul64Ex [▶ 83]	Multiplies two 64 bit numbers. Result is 64 bit number (with overflow check).
UInt64Div64 [▶ 83]	Division of one variable by another
UInt64Div64Ex [▶ 84]	Division of one variable by another (including remainder)
UInt64Mod64 [▶ 85]	Modulo division of one variable by another
UInt64And [▶ 86]	Bitwise AND of operands
UInt64Or [▶ 86]	Bitwise OR of operands
UInt64Not [▶ 87]	Bitwise NOT of operand
UInt64Xor [▶ 87]	Bitwise XOR of operands
UInt64Rol [▶ 90]	Bitwise rotation of an operand to the left
UInt64Ror [▶ 90]	Bitwise rotation of an operand to the right
UInt64Shl [▶ 89]	Bitwise left-shift of an operand
UInt64Shr [▶ 89]	Bitwise right-shift of an operand
UInt64Min [▶ 87]	Minimum function
UInt64Max [▶ 88]	Maximum function
UInt64Limit [▶ 88]	Limiting function
UInt64isZero [▶ 91]	Checks if value of 64 bit integer is zero
UINT64_TO_STRING [▶ 92]	Converts 64 bit number to STRING
UINT64_TO_LREAL [▶ 92]	Converts 64 bit number to LREAL
STRING_TO_UINT64 [▶ 91]	Converts STRING to 64 bit number
LREAL_TO_UINT64 [▶ 91]	Converts LREAL to 64 bit number

64 bit functions (signed)

Name	Description
LARGE_INTEGER [▶ 93]	Initializes/sets 64 bit number value

Name	Description
Int64Add64 [▶ 93]	Adds two 64 bit numbers
Int64Add64Ex [▶ 94]	Adds two 64 bit numbers (with overflow check)
Int64Sub64 [▶ 94]	Subtracts two 64 bit numbers
Int64Cmp64 [▶ 95]	Compares two 64 bit numbers
Int64Div64Ex [▶ 94]	Division of one variable by another (including remainder)
Int64Not [▶ 95]	Bitwise NOT of operand
Int64isZero [▶ 96]	Checks if value of 64 bit integer is zero
Int64Negate [▶ 96]	Negates 64 bit number
INT64_TO_LREAL [▶ 97]	Converts 64 bit number to LREAL
LREAL_TO_INT64 [▶ 96]	Converts LREAL to 64 bit number
LARGE_TO_ULARGE [▶ 98]	Converts signed 64 bit number to unsigned 64 bit number
ULARGE_TO_LARGE [▶ 97]	Converts unsigned 64 bit number to signed 64 bit number

16 bit fixed point (signed)

Name	Description
FIX16Add [▶ 100]	Adds two fixed point numbers
FIX16Align [▶ 101]	Changes the resolution of fixed point number
FIX16Sub [▶ 103]	Subtracts two fixed point numbers
FIX16Div [▶ 102]	Division of two fixed point numbers
FIX16Mul [▶ 103]	Multiplies two fixed point numbers
LREAL_TO_FIX16 [▶ 98]	Converts LREAL to fixed point number
WORD_TO_FIX16 [▶ 99]	Converts WORD to fixed point number
FIX16_TO_LREAL [▶ 99]	Converts fixed point number to LREAL
FIX16_TO_WORD [▶ 100]	Converts fixed point number to WORD

Byte order converting functions

Name	Description
HOST_TO_BE16 [▶ 122]	Host-to-network converting (16 bit number)
HOST_TO_BE32 [▶ 123]	Host-to-network converting (32 bit number)
HOST_TO_BE64 [▶ 123]	Host-to-network converting (64 bit number)
HOST_TO_BE128 [▶ 124]	Host-to-network converting (128 bit number)
BE16_TO_HOST [▶ 124]	Network-to-host converting (16 bit number)
BE32_TO_HOST [▶ 124]	Network-to-host converting (32 bit number)
BE64_TO_HOST [▶ 125]	Network-to-host converting (64 bit number)
BE128_TO_HOST [▶ 125]	Network-to-host converting (128 bit number)

Other functions

Name	Description
FB_BasicPID [▶ 153]	Simple PID controller
F_GetVersionTcUtilities [▶ 74]	Returns library version info
IsFinite [▶ 51]	Checks if REAL or LREAL variable can be represented in Institute of Electrical and Electronics Engineers (IEEE) format.
F_YearIsLeapYear [▶ 60]	Determines whether the given year is a leap year.
F_GetMaxMonthDays [▶ 58]	Returns the number of days in a month.

Name	Description
F_GetDOYOfYearMonthDay [▶ 59]	Calculates the day number of the year
F_GetMonthOfDOY [▶ 59]	Calculates the month from day number of the year
F_GetDayOfWeek [▶ 66]	Returns the day number of a week.
F_GetWeekOfTheYear [▶ 66]	Returns the week number of the year.
F_GetDayOfMonthEx [▶ 61]	Calculates the day of month of specific month, month's week and year
RTC [▶ 212]	RTC (Real Time Clock)
RTC_EX [▶ 213]	RTC (Real Time Clock)
RTC_EX2 [▶ 214]	RTC (Real Time Clock)
FB_FileRingBuffer [▶ 166]	Writes/reads datasets to/from file (FIFO).
FB_MemRingBuffer [▶ 169]	Writes/reads datasets to/from memory buffer (FIFO).
FB_MemRingBufferEx [▶ 170]	Writes/reads datasets to/from memory buffer (FIFO).
FB_StringRingBuffer [▶ 176]	Writes/reads strings to/from memory buffer (FIFO).
FB_MemStackBuffer [▶ 174]	Writes/reads datasets to/from memory buffer (LIFO).
FB_HashTableCtrl [▶ 190], F_CreateHashTableHnd [▶ 67]	Simple hash table.
FB_LinkedListCtrl [▶ 192], F_CreateLinkedListHnd [▶ 68]	Simple linked list (doubly-linked).
DCF77_TIME [▶ 221]	Simple DCF77 decoder.
DCF77_TIME_EX [▶ 224]	DCF77 decoder with plausibility check of two telegrams and time zone information.

CSV format helper

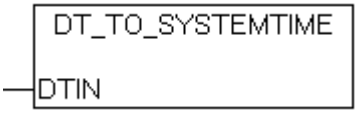
Name	Description
CSVFIELD_TO_STRING [▶ 69]	Converts string containing data field in CSV format to PLC string value
STRING_TO_CSVFIELD [▶ 71]	Converts PLC string value to string containing data field in CSV format
CSVFIELD_TO_ARG [▶ 70]	Converts buffer containing data field in CSV format to any PLC variable value
ARG_TO_CSVFIELD [▶ 72]	Converts any PLC variable value to buffer containing data field in CSV format
FB_CSVMemBufferReader [▶ 227]	Splits memory buffer containing data records in CSV format to single data fields
FB_CSVMemBufferWriter [▶ 228]	Creates data records in CSV format in memory buffer using single data fields

Also see about this

- TC_CpuUsage [▶ 196]
- HEXSTR_TO_DATA [▶ 50]

3 Functions

3.1 DT_TO_SYSTEMTIME



The "DT_TO_SYSTEMTIME" function allows a PLC variable in DATE_AND_TIME format (DT) to be converted to a Windows system time structure. The system time has a resolution of 1ms, while the resolution of DATE_AND_TIME is 1s. The "wMilliseconds" variable in the system time structure therefore always returns the value zero.

FUNCTION DT_TO_SYSTEMTIME : TIMESTRUCT

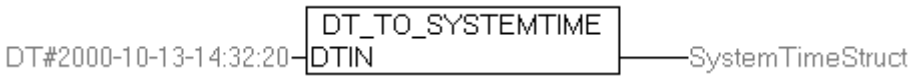
TIMESTRUCT [▶ 230]

```
VAR_INPUT
    DTIN        : DT;
END_VAR
```

DTIN:: The date and time to be converted, in DATE_AND_TIME format.

Example of a call in FBD:

```
PROGRAM SystemTimeTest
VAR
    SystemTimeStruct  :TIMESTRUCT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 and above	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.2 DT_TO_FILETIME

The function "DT_TO_FILETIME" allows to convert a PLC variable in DATE_AND_TIME format (DT) into the FILETIME format (64bit).

FUNCTION DT_TO_FILETIME : T_FILETIME

T_FILETIME [▶ 239]

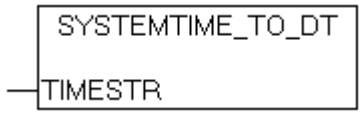
```
VAR_INPUT
    DTIN        : DT;
END_VAR
```

DTIN:: Date and time to read in DATE_AND_TIME format.

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 and higher TwinCAT v2.10.0 Build > 1232 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.3 SYSTEMTIME_TO_DT



The "SYSTEMTIME_TO_DT" function allows the Windows system time structure to be converted to the DATE_AND_TIME format (DT) usual in a PLC. The system time has a resolution of 1ms, while the resolution of DATE_AND_TIME is 1s. The milliseconds from the system time are used in the course of the conversion to determine the direction of rounding for the returned DATE_AND_TIME value. Set the wMilliseconds member of the Windows system time structure to zero to deactivate the rounding.

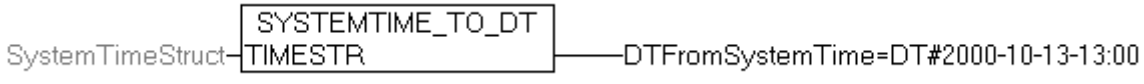
FUNCTION SYSTEMTIME_TO_DT : DT

```
VAR_INPUT
    TIMESTR     : TIMESTRUCT;
END_VAR
```

TIMESTR: The structure [▶ 230] with the Windows system time requiring conversion.

Example of a call in FBD:

```
PROGRAM SystemTimeTest
VAR
    SystemTimeStruct  :TIMESTRUCT;
    DTFromSystemTime  :DT;
END_VAR
```

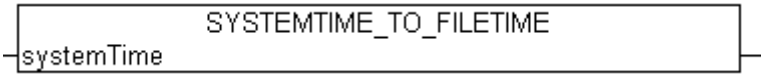


Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 and above	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Development environment	Target system type	PLC libraries to include
		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.4 SYSTEMTIME_TO_FILETIME



The "SYSTEMTIME_TO_FILETIME" function allows the Windows system time structure to be converted to the file time format. The wDayOfWeek member of the systemTime variable is ignored. The system time year must be greater than 1601 and cannot be greater than 30827.

FUNCTION SYSTEMTIME_TO_FILETIME : T_FILETIME

T_FILETIME [▶ 239]

```
VAR_INPUT
    systemTime : TIMESTRUCT;
END_VAR
```

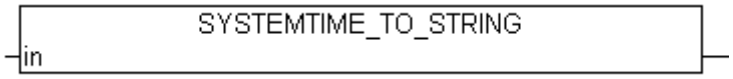
systemTime: The structure [▶ 230] with the Windows system time requiring conversion.

Return value	Description
0	Invalid systemTime parameter.
> 0	64 bit file time.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.5 SYSTEMTIME_TO_STRING



The "SYSTEMTIME_TO_STRING" function converts Windows sytem time struct to string. The string format is: **YYYY-MM-DD-hh:mm:ss.xxx**, where:

- YYYY: Year (1601..9999);
- MM: Month (01..12);
- DD: Day (01..31);
- hh: Hour (00..23);
- mm: Minute (00..59);
- ss: Second (00..59)
- xxx: Millisecond (000..999)

FUNCTION SYSTEMTIME_TO_STRING : STRING(24)

```
VAR_INPUT
    in              :TIMESTRUCT;
END_VAR
```

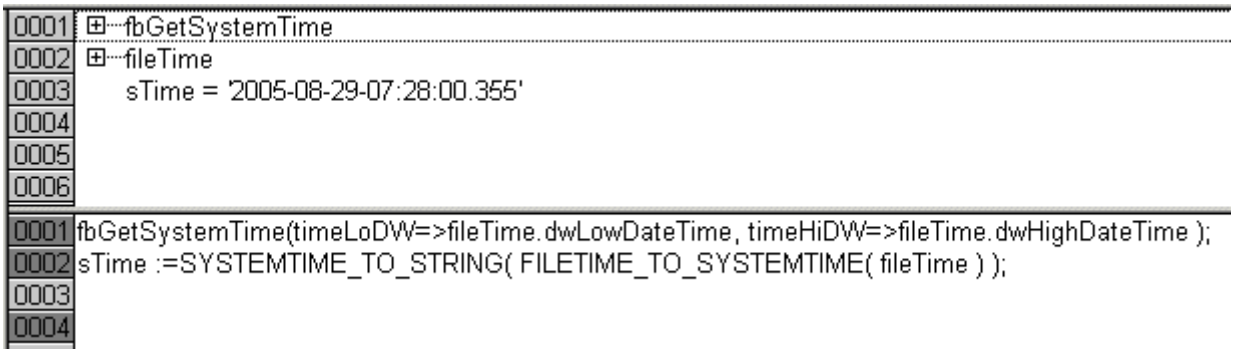
in: The structure [▶ 230] with the Windows system time requiring for the conversion.

Sample in ST:

```
PROGRAM MAIN
VAR
    fbGetSystemTime  : GETSYSTEMTIME;
    fileTime         : T_FILETIME;
    sTime            : STRING;
END_VAR
```

```
fbGetSystemTime(timeLoDW=>fileTime.dwLowDateTime, timeHiDW=>fileTime.dwHighDateTime );
sTime :=SYSTEMTIME_TO_STRING( FILETIME_TO_SYSTEMTIME( fileTime ) );
```

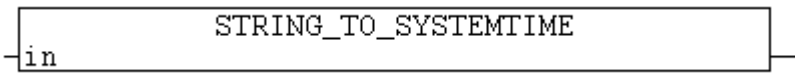
Online view:



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1241	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.6 STRING_TO_SYSTEMTIME



The function converts string to Windows system time struct.

FUNCTION STRING_TO_SYSTEMTIME: TIMESTRUCT

TIMESTRUCT [▶ 230]

VAR_INPUT

```
VAR_INPUT
    in : STRING(23);
END_VAR
```

in: The string format is: **YYYY-MM-DD-hh:mm:ss.xxx**, where:

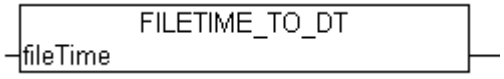
- YYYY: Year (1601.. 9999);
- MM: Month (01..12);
- DD: Day (01..31);

- hh: Hour (00..23);
- mm: Minute (00..59);
- ss: Second (00..59);
- xxx: Millisecond (000..999);

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1310 or higher	PC or CX (x86, ARM)	TcUtilities.Lib

3.7 FILETIME_TO_DT



The function "FILETIME_TO_DT" converts time from FILETIME format into DATE_AND_TIME format (DT). The DT format has a smaller value range than the FILETIME format and only an accuracy of seconds. On this account the FILETIME value to convert is limited. The allowed minimum conforms to the value *DT#1970-01-01-00:00:00* and the maximum to *DT#2106-02-06-06:28:15*.

FUNCTION FILETIME_TO_DT : DT

```
VAR_INPUT
    fileTime        : T_FILETIME;
END_VAR
```

fileTime: Time to convert in FILETIME format [▶ 239].

Example in ST:

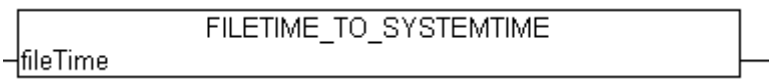
```
PROGRAM MAIN
VAR
    fbSystemTime    : GETSYSTEMTIME;
    timeAsFileTime  : T_FILETIME;
    timeAsDT    : DT;
END_VAR
```

```
fbSystemTime( timeLoDW =>timeAsFileTime.dwLowDateTime , timeHiDW =>timeAsFileTime.dwHighDateTime );
timeAsDT := FILETIME_TO_DT( timeAsFileTime );
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 and higher TwinCAT v2.10.0 Build > 1240 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.8 FILETIME_TO_SYSTEMTIME



The function "FILETIME_TO_SYSTEMTIME" converts time from FILETIME format into the "readable" SYSTEMTIME format. The conversion fails if the most significant bit of 64 bit file Time variables is set. In this case the TIMESTRUCT member variables have the value zero.

FUNCTION FILETIME_TO_SYSTEMTIME: TIMESTRUCT

TIMESTRUCT [▶ 230]

```
VAR_INPUT
    fileTime        : T_FILETIME;
END_VAR
```

fileTime:: Time to convert in FILETIME format [▶ 239]

Example in ST:

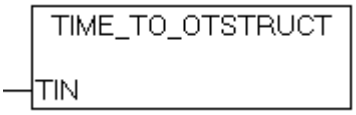
```
PROGRAM MAIN
VAR
    fbSystemTime     : GETSYSTEMTIME;
    timeAsFileTime   : T_FILETIME;
    timeAsSystemTime : TIMESTRUCT;
END_VAR
```

```
(* program code *)
fbSystemTime( timeLoDW =>timeAsFileTime.dwLowDateTime , timeHiDW =>timeAsFileTime.dwHighDateTime );
timeAsSystemTime := FILETIME_TO_SYSTEMTIME( timeAsFileTime );
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 and higher TwinCAT v2.10.0 Build > 1240 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.9 TIME_TO_OTSTRUCT



The "TIME_TO_OTSTRUCT" function allows a PLC variable in TIME format to be converted to a structure with milliseconds, seconds, minutes, hours, days and weeks.

FUNCTION TIME_TO_OTSTRUCT : OTSTRUCT

OTSTRUCT [▶ 230]

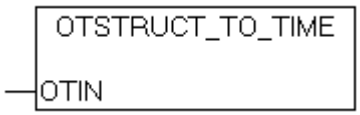
```
VAR_INPUT
    TIN         :TIME;
END_VAR
```

TIN: The value to be converted.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 and above	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.10 OTSTRUCT_TO_TIME



The "OTSTRUCT_TO_TIME" function allows the structure with milliseconds, seconds, minutes, hours, days and weeks to be converted to the TIME format usual in a PLC.

FUNCTION OTSTRUCT_TO_TIME : TIME

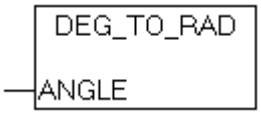
```
VAR_INPUT
    OTIN        : OTSTRUCT;
END_VAR
```

OTIN: The value [▶ 230] to be converted.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 and above	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.11 DEG_TO_RAD



Converts degrees to radiants.

FUNCTION DEG_TO_RAD : LREAL

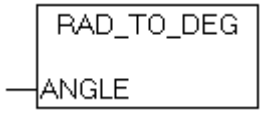
```
VAR_INPUT
    ANGLE       : LREAL;
END_VAR
```

ANGLE:: Value to convert.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.12 RAD_TO_DEG



Converts radiants to degrees.

FUNCTION RAD_TO_DEG : LREAL

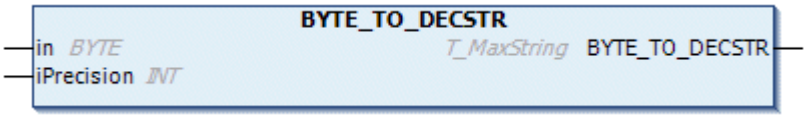
```
VAR_INPUT
    ANGLE:      LREAL;
END_VAR
```

ANGLE :: Value to convert.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.13 BYTE_TO_DECSTR



This function converts a decimal number into a decimal string (base 10).

FUNCTION BYTE_TO_DECSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : BYTE;
    iPrecision  : INT;
END_VAR
```

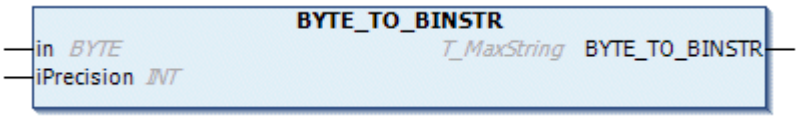
in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.14 BYTE_TO_BINSTR



This function converts a decimal number into a binary string (base 2).

FUNCTION BYTE_TO_BINSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : BYTE;
    iPrecision  : INT;
END_VAR
```

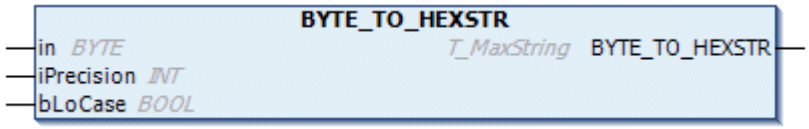
in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.15 BYTE_TO_HEXSTR



This function converts a decimal number into a hexadecimal string (base 16).

FUNCTION BYTE_TO_HEXSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : BYTE;
    iPrecision  : INT;
    bLoCase     : BOOL;
END_VAR
```

in: The decimal number requiring conversion.

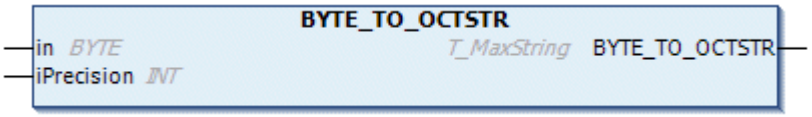
iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

bLoCase: This parameter determines whether lower- or upper-case letters are used in the conversion. FALSE => "ABCDEF", TRUE => "abcdef".

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.16 BYTE_TO_OCTSTR



This function converts a decimal number into an octal string (base 8).

FUNCTION BYTE_TO_OCTSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : BYTE;
    iPrecision  : INT;
END_VAR
```

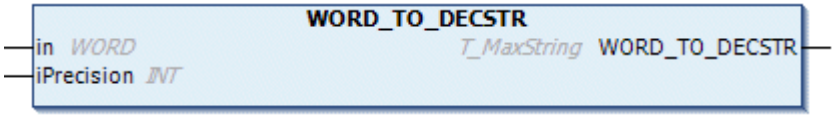
in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.17 WORD_TO_DECSTR



This function converts a decimal number into a decimal string (base 10).

FUNCTION WORD_TO_DECSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : WORD;
    iPrecision  : INT;
END_VAR
```

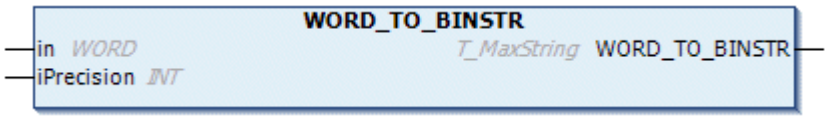
in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.18 WORD_TO_BINSTR



This function converts a decimal number into a binary string (base 2).

FUNCTION WORD_TO_BINSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : WORD;
    iPrecision  : INT;
END_VAR
```

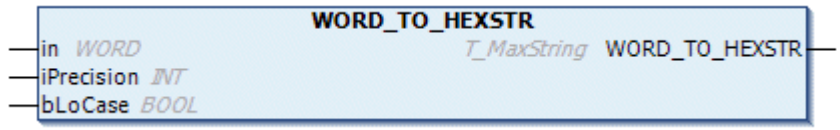
in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.19 WORD_TO_HEXSTR



This function converts a decimal number into a hexadecimal string (base 16).

FUNCTION WORD_TO_HEXSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : WORD;
    iPrecision  : INT;
    bLoCase     : BOOL;
END_VAR
```

in: The decimal number requiring conversion.

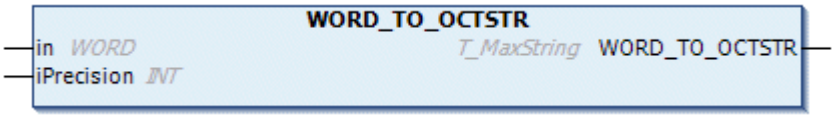
iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

bLoCase: This parameter determines whether lower or upper case letters are used in the conversion. FALSE => "ABCDEF", TRUE => "abcdef".

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.20 WORD_TO_OCTSTR



This function converts a decimal number into an octal string (base 8).

FUNCTION WORD_TO_OCTSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : WORD;
    iPrecision  : INT;
END_VAR
```

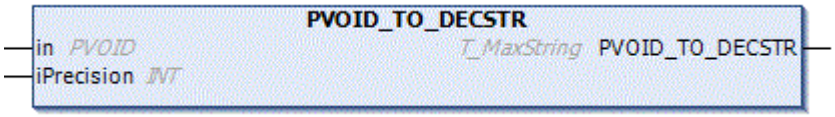
in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.21 PVOID_TO_DECSTR



The function converts the value of a pointer variable of type PVOID into a decimal string (basis 10).

FUNCTION PVOID_TO_DECSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : PVOID;
    iPrecision  : INT;
END_VAR
```

in: The pointer variable to be converted.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Example:

```
PROGRAM MAIN
VAR
    s1   : STRING;
    s2   : STRING;
    s3   : STRING;
    s4   : STRING;
    s5   : STRING;
    s6   : STRING;
    nCnt : WORD;
    pCnt : PVOID := 0;
END_VAR
```

```
pCnt := 0;s1   := PVOID_TO_DECSTR( pCnt, 0 );
s2   := PVOID_TO_DECSTR( pCnt, 1 );
s3   := PVOID_TO_DECSTR( pCnt, 16 );
pCnt := ADR( nCnt );
s4   := PVOID_TO_DECSTR( pCnt, 0 );
s5   := PVOID_TO_DECSTR( pCnt, 1 );
s6   := PVOID_TO_DECSTR( pCnt, 16 );
```

The result:

s1 = ''

s2 = '0'

s3 = '0000000000000000'

s4 = '2279473749' (may vary)

s5 = '2279473749' (may vary)

s6 = '0000002279473749' (may vary)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

Development environment	Target system type	PLC libraries to be linked
		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.22 PVOID_TO_BINSTR

The function converts the value of a pointer variable of type PVOID into a binary string (basis 2).

FUNCTION PVOID_TO_BINSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : PVOID;
    iPrecision  : INT;
END_VAR
```

in: The pointer variable to be converted.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Example:

```
PROGRAM MAIN
VAR
    s1   : STRING;
    s2   : STRING;
    s3   : STRING;
    s4   : STRING;
    s5   : STRING;
    s6   : STRING;
    nCnt : BYTE;
    pCnt : PVOID := 0;
END_VAR
```

```
pCnt := 0;s1   := PVOID_TO_BINSTR( pCnt, 0 );s2   := PVOID_TO_BINSTR( pCnt, 1 );s3   := PVOID_TO_BIN
STR( pCnt, 32 );pCnt := ADR( nCnt );s4   := PVOID_TO_BINSTR( pCnt, 0 );s5   := PVOID_TO_BINSTR( pCnt
, 1 );s6   := PVOID_TO_BINSTR( pCnt, 32 );
```

The result:

s1 = ''

s2 = '0'

s3 = '00000000000000000000000000000000'

s4 = '10000111110111100000001001010101' (may vary)

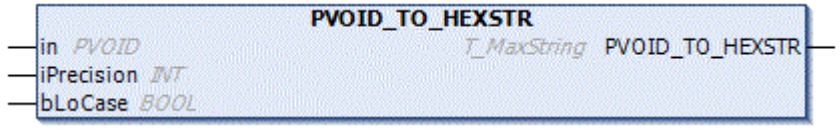
s5 = '10000111110111100000001001010101' (may vary)

s6 = '10000111110111100000001001010101' (may vary)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.23 PVOID_TO_HEXSTR



The function converts the value of a pointer variable of type PVOID into a hexadecimal string (basis 16).

FUNCTION PVOID_TO_HEXSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : PVOID;
    iPrecision  : INT;
    bLoCase     : BOOL;
END_VAR
```

in: The pointer variable to be converted.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

bLoCase: This parameter determines whether lower- or upper-case letters are used in the conversion. FALSE => "ABCDEF", TRUE => "abcdef".

Example:

```
PROGRAM MAIN
VAR
    s1   : STRING;
    s2   : STRING;
    s3   : STRING;
    s4   : STRING;
    s5   : STRING;
    s6   : STRING;
    s7   : STRING;
    s8   : STRING;
    s9   : STRING;
    s10  : STRING;
    s11  : STRING;
    s12  : STRING;
    nCnt : WORD;
    pCnt : PVOID := 0;
END_VAR
```

```
pCnt := 0;
s1   := PVOID_TO_HEXSTR( pCnt, 0, FALSE );
s2   := PVOID_TO_HEXSTR( pCnt, 0, TRUE );
s3   := PVOID_TO_HEXSTR( pCnt, 1, FALSE );
s4   := PVOID_TO_HEXSTR( pCnt, 1, TRUE );
s5   := PVOID_TO_HEXSTR( pCnt, 16, FALSE );
s6   := PVOID_TO_HEXSTR( pCnt, 16, TRUE );
pCnt := ADR( nCnt );
s7   := PVOID_TO_HEXSTR( pCnt, 0, FALSE );
s8   := PVOID_TO_HEXSTR( pCnt, 0, TRUE );
```

```
s9   := PVOID_TO_HEXSTR( pCnt, 1, FALSE );
s10  := PVOID_TO_HEXSTR( pCnt, 1, TRUE );
s11  := PVOID_TO_HEXSTR( pCnt, 16, FALSE );
s12  := PVOID_TO_HEXSTR( pCnt, 16, TRUE );
```

The result:

s1 = ''

s2 = ''

s3 = '0'

s4 = '0'

s5 = '0000000000000000'

s6 = '0000000000000000'

s7 = '87CBC255' (may vary)

s8 = '87cbc255' (may vary)

s9 = '87CBC255' (may vary)

s10 = '87cbc255' (may vary)

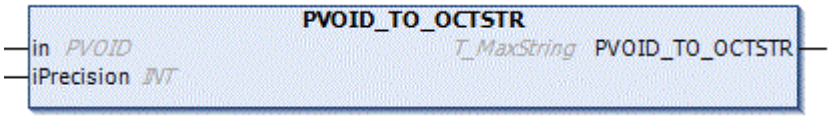
s11 = '0000000087CBC255' (may vary)

s12 = '0000000087cbc255' (may vary)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.24 PVOID_TO_OCTSTR



The function converts the value of a pointer variable of type PVOID into an octal string (basis 8).

FUNCTION PVOID_TO_OCTSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : PVOID;
    iPrecision  : INT;
END_VAR
```

in: The pointer variable to be converted.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Example:

```
PROGRAM MAIN
VAR
    s1   : STRING;
    s2   : STRING;
    s3   : STRING;
    s4   : STRING;
    s5   : STRING;
    s6   : STRING;
    nCnt : WORD;
    pCnt : PVOID := 0;
END_VAR
```

```
pCnt := 0;
s1   := PVOID_TO_OCTSTR( pCnt, 0 );
s2   := PVOID_TO_OCTSTR( pCnt, 1 );
s3   := PVOID_TO_OCTSTR( pCnt, 16 );
pCnt := ADR( nCnt );
s4   := PVOID_TO_OCTSTR( pCnt, 0 );
s5   := PVOID_TO_OCTSTR( pCnt, 1 );
s6   := PVOID_TO_OCTSTR( pCnt, 16 );
```

The result:

s1 = ''

s2 = '0'

s3 = '0000000000000000'

s4 = '20767501125' (may vary)

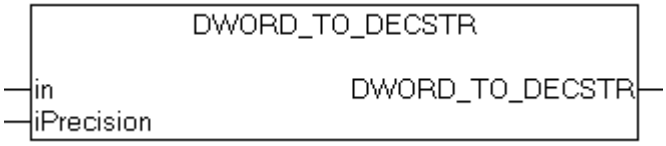
s5 = '20767501125' (may vary)

s6 = '0000020767501125' (may vary)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.25 DWORD_TO_DECSTR



This function converts a decimal number into a decimal string (base 10).

FUNCTION DWORD_TO_DECSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : DWORD;
    iPrecision  : INT;
END_VAR
```

in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Example in ST:

```
PROGRAM MAIN
VAR
    s1      : STRING;
    s2      : STRING;
    s3      : STRING;
    nCnt    : WORD;
END_VAR
```

```
nCnt := 43981;
s1 := DWORD_TO_DECSTR( nCnt, 1);
s2 := DWORD_TO_DECSTR(  nCnt, 10 );
nCnt := 0;
s3 := DWORD_TO_DECSTR( nCnt, 0 );
```

The result:

s1 = '43981'

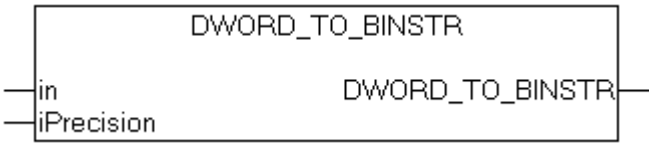
s2 = '0000043981'

s3 = ''

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.26 DWORD_TO_BINSTR



This function converts a decimal number into a binary string (base 2).

FUNCTION DWORD_TO_BINSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : DWORD;
    iPrecision  : INT;
END_VAR
```

in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Example in ST:

```
PROGRAM MAIN
VAR
    s1      : STRING;
    s2      : STRING;
    s3      : STRING;
    nCnt    : BYTE;
END_VAR
```

```
s1 := DWORD_TO_BINSTR( 16#81, 16 );
nCnt := 15;
s2 := DWORD_TO_BINSTR( nCnt, 1 );
nCnt := 0;
s3 := DWORD_TO_BINSTR( nCnt, 0 );
```

The result:

s1 = '0000000010000001'

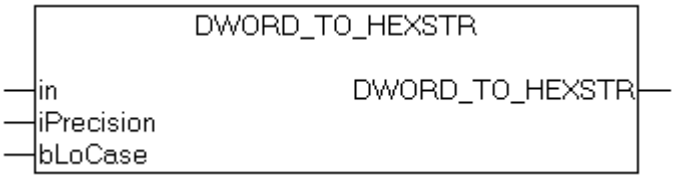
s2 = '1111'

s3 = ''

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.27 DWORD_TO_HEXSTR



This function converts a decimal number into a hexadecimal string (base 16).

FUNCTION DWORD_TO_HEXSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : DWORD;
    iPrecision  : INT;
    bLoCase     : BOOL;
END_VAR
```

in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

bLoCase: This parameter determines whether lower- or upper-case letters are used in the conversion. FALSE => "ABCDEF", TRUE => "abcdef".

Example in ST:

```
PROGRAM MAIN
VAR
    s1      : STRING;
    s2      : STRING;
    s3      : STRING;
    s4      : STRING;
    nCnt    : WORD;
END_VAR
```

```
nCnt := 43981;
s1 := DWORD_TO_HEXSTR( nCnt, 1, FALSE );
s2 := DWORD_TO_HEXSTR( nCnt, 1, TRUE );
nCnt := 15;
s3 := DWORD_TO_HEXSTR(  nCnt, 4, FALSE );
nCnt := 0;
s4 := DWORD_TO_HEXSTR( nCnt, 0, FALSE );
```

The result:

s1 = 'ABCD'

s2 = 'abcd'

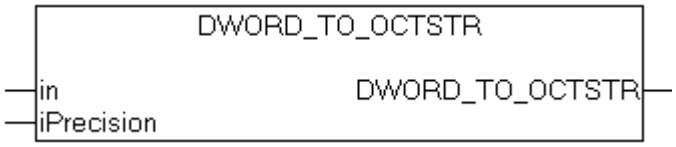
s3 = '000F'

s4 = ''

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.28 DWORD_TO_OCTSTR



This function converts a decimal number into an octal string (base 8).

FUNCTION DWORD_TO_OCTSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : DWORD;
    iPrecision  : INT;
END_VAR
```

in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty.

Example in ST:

```
PROGRAM MAIN
VAR
    s1      : STRING;
    s2      : STRING;
```

```
    s3      : STRING;
    nCnt    : WORD;
END_VAR
```

```
nCnt := 43981;
s1 := DWORD_TO_OCTSTR( nCnt, 1);
s2 := DWORD_TO_OCTSTR(  nCnt, 10 );
nCnt := 0;
s3 := DWORD_TO_OCTSTR( nCnt, 0 );
```

The result:

s1 = '125715'

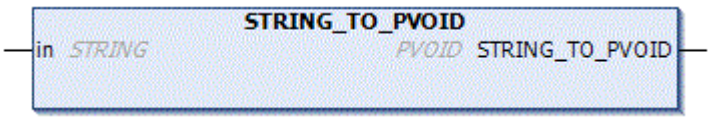
s2 = '0000125715'

s3 = ''

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.29 STRING_TO_PVOID



The function converts a string variable into a pointer variable of type PVOID. The function returns zero if the input string contains incorrect characters and cannot be interpreted as an address.

FUNCTION STRING_TO_PVOID : PVOID

VAR_INPUT

```
VAR_INPUT
    in : STRING;
END_VAR
```

in:String variable to be converted.

Example:

```
PROGRAM MAIN
VAR
    sP1 : STRING := '16#89345678';
    sP2 : STRING := '8#21115053170';
    sP3 : STRING := '2#10001001001101000101011001111000';
    sP4 : STRING := '2301908600';
    sP5 : STRING := '';
    pP1 : PVOID := 0;
    pP2 : PVOID := 0;
    pP3 : PVOID := 0;
    pP4 : PVOID := 0;
    pP5 : PVOID := 0;
END_VAR
```

```
pP1 := STRING_TO_PVOID( sP1 );
pP2 := STRING_TO_PVOID( sP2 );
pP3 := STRING_TO_PVOID( sP3 );
pP4 := STRING_TO_PVOID( sP4 );
pP5 := STRING_TO_PVOID( sP5 );
```

The result:

pP1 = 2301908600

pP2 = 2301908600

pP3 = 2301908600

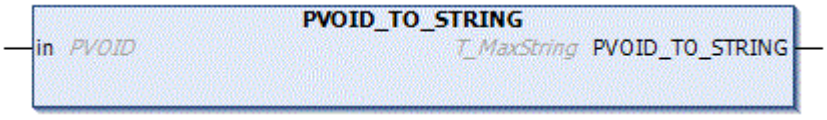
pP4 = 2301908600

pP5 = 0

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.30 PVOID_TO_STRING



The function converts the value of a pointer variable of type PVOID into a hexadecimal string (basis 16). The hexadecimal string has the PLC prefix: '16#'. The resolution is fixed at 8 digits in a 32-bit system.

FUNCTION PVOID_TO_STRING : T_MaxString

T_MaxString

```
VAR_INPUT
    in      : PVOID;
END_VAR
```

in: The pointer variable to be converted.

Example:

```
PROGRAM MAIN
VAR
    s1   : STRING;
    s2   : STRING;
    nCnt : BYTE;
    p1   : POINTERTOBYTE := 0;
    p2   : POINTERTOBYTE := ADR( nCnt );
END_VAR
```

```
s1 := PVOID_TO_STRING( p1 );
s2 := PVOID_TO_STRING( p2 );
```

The result on a 32-bit system:

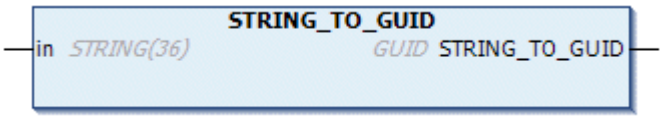
s1 = '16#00000000'

s2 = "16#87DE0255' (may vary)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.31 STRING_TO_GUID



This function converts a registry GUID string variable (without curly brackets) into a structured GUID [▶ 248] variable.

FUNCTION STRING_TO_GUID : GUID

GUID [▶ 248]

VAR_INPUT

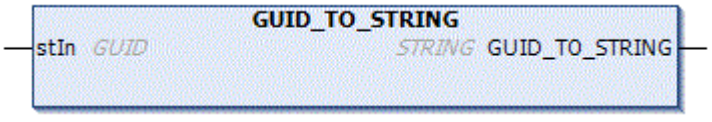
```
VAR_INPUT
    in : STRING(36);
END_VAR
```

Return value	Meaning
'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'	No error ('x' is a hexadecimal half-byte)
'00000000-0000-0000-0000-000000000000'	No error, GUID has the initial value (all bytes are zero)

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.32 GUID_TO_STRING



This function converts a structured GUID [▶ 248] variable into a GUID string variable (without curly brackets).

FUNCTION GUID_TO_STRING : STRING

VAR_INPUT

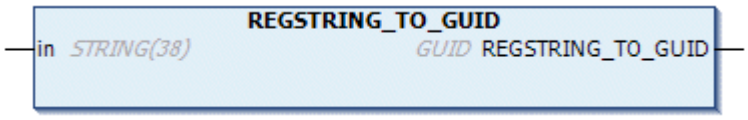
```
VAR_INPUT
    stIn : GUID;
END_VAR
```

Return value	Meaning
'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'	No error (‘x’ is a hexadecimal half-byte)
00000000-0000-0000-0000-000000000000'	No error, GUID has the initial value (all bytes are zero)

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.33 REGSTRING_TO_GUID



This function converts a registry GUID string variable (enclosed in curly brackets) into a structured GUID [▶ 248] variable.

FUNCTION REGSTRING_TO_GUID: GUID

GUID [▶ 248]

VAR_INPUT

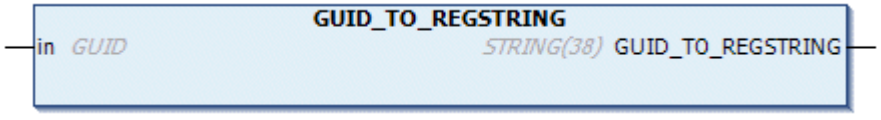
```
VAR_INPUT
      in : STRING(38);
END_VAR
```

Return value	Meaning
'{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'	No error (‘x’ is a hexadecimal half-byte).
'{00000000-0000-0000-0000-000000000000}'	Conversion failed or GUID has initial value (all bytes are zero)

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.34 GUID_TO_REGSTRING



This function converts a structured GUID [▶ 248] variable into a registry GUID string variable (enclosed in curly brackets).

FUNCTION GUID_TO_REGSTRING : STRING(38)

VAR_INPUT

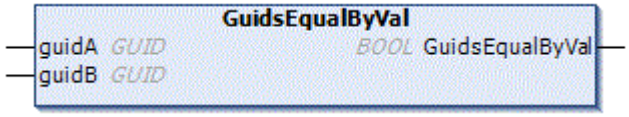
```
VAR_INPUT
    in      : GUID;
END_VAR
```

Return value	Meaning
'{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'	No error ('x' is a hexadecimal half-byte)
'{00000000-0000-0000-0000-000000000000}'	No error, GUID has the initial value (all bytes are zero)

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.35 GuidsEqualByVal



This function compares two GUID values.

FUNCTION GuidsEqualByVal : BOOL

VAR_INPUT

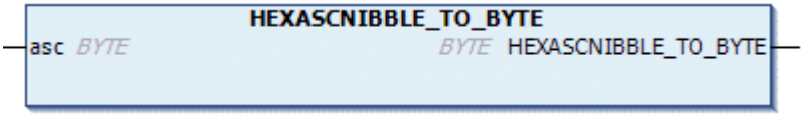
```
VAR_INPUT
    guidA : GUID;
    guidB : GUID;
END_VAR
```

Return value	Meaning
FALSE	guidA <> guidB
TRUE	guidA = guidB

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.36 HEXASCNIBBLE_TO_BYTE



This function converts the ASCII code of a hexadecimal half-byte character into the decimal value.

FUNCTION HEXASCNIBBLE_TO_BYTE : BYTE

VAR_INPUT

```
VAR_INPUT
    asc     : BYTE;
END_VAR
```

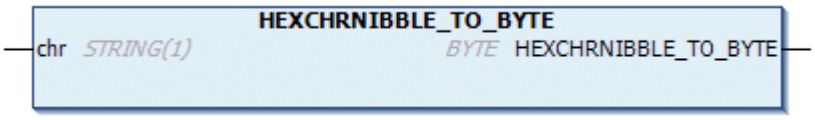
asc: Ascii-code of a hexadecimal half-byte character (Ascii code from: ‘0’ to ‘9’ or ‘a’ to ‘f’ or ‘A’ to ‘F’).

Return value	Meaning
0 to 15	Successful, no error.
255	Error, wrong input parameter value.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.37 HEXCHRNIBBLE_TO_BYTE



This function converts a hexadecimal half-byte character into its decimal value.

FUNCTION HEXCHRNIBBLE_TO_BYTE : BYTE

VAR_INPUT

```
VAR_INPUT
    chr     : STRING(1);
END_VAR
```

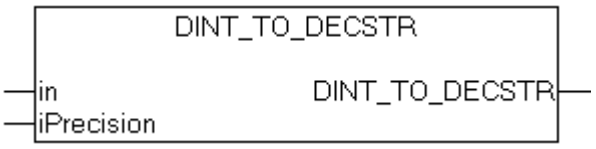
chr: Hexadecimal half-byte character (‘0’ to ‘9’ or ‘a’ to ‘f’ or ‘A’ to ‘F’).

Return value	Meaning
0 to 15	Successful, no error.
255	Error, wrong input parameter value.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.38 DINT_TO_DECSTR



This function converts a signed decimal number into a decimal string (base 10).

FUNCTION DINT_TO_DECSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    in          : DINT;
    iPrecision  : INT;
END_VAR
```

in: The decimal number requiring conversion.

iPrecision: Minimum number of displayed digits (digits). If the actual number of significant digits is less than the *iPrecision* parameter, the resulting string is filled with zeros from the left. If the number of significant digits is greater than the *iPrecision* parameter, the resulting string is not cut off! If the *iPrecision* parameter and the *in* parameter are zero, the resulting string is empty. For negative numbers, the negative sign will appear in the resulting string.

Example in ST:

```
PROGRAM MAIN
VAR
    s1      : STRING;
    s2      : STRING;
    s3      : STRING;
    s4      : STRING;
    iCnt    : INT;
END_VAR
```

```
iCnt := -1234;
s1 := DINT_TO_DECSTR( iCnt, 1);
s2 := DINT_TO_DECSTR(  iCnt, 10 );
iCnt := 0;
s3 := DINT_TO_DECSTR( iCnt, 0 );
iCnt := 1234;
s4 := DINT_TO_DECSTR( iCnt, 10 );
```

The result:

s1 = '-1234'

s2 = '-0000001234'

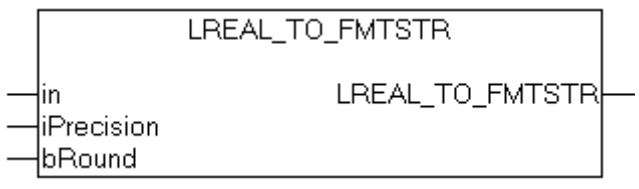
s3 = ''

s4 = '0000001234'

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.39 LREAL_TO_FMTSTR



The function converts and formats a floating-point number into a string variable with the following format: **[–]dddd.dddd** (dddd are decimal numbers). The number of digits before the decimal point depends on the value of the floating-point number. The number of digits after the decimal point depends on the required precision. The sign only appears for negative values. **'#INF'** is returned for infinite positive values, **'-#INF'** for infinite negative values. If the variable transferred has an illegal value (NaN, Not-a-Number), **'#QNAN'** or **'-#QNAN'** is returned. If the length of the formatted string exceeds the maximum permissible length of the resulting string, **'#OVF'** or **'-#OVF'** is returned.

FUNCTION LREAL_TO_FMTSTR : STRING(510)

```
VAR_INPUT
    in          : LREAL;
    iPrecision  : INT;
    bRound      : BOOL;
END_VAR
```

in: floating point number that is to be converted and formatted.

iPrecision: precision. The value determines the number of digits after the decimal point. With the minimum value (zero), no decimal places are displayed. The maximum value of *iPrecision* is limited by the number of digits before the decimal point and the maximum permissible length of the resulting string. If *in* = 0 and *iPrecision* = 0 then string '0' is returned.

bRound: if the *bRound* parameter is set, the formatted string is rounded to the respective number of decimal places (*iPrecision*). The following rule applies for rounding: if the decimal number after the last required decimal place is >= 5, the value is rounded up, otherwise not.

Sample:

0.46523 is to be converted into a string with two decimal places and rounded.

```
sOut := LREAL_TO_FMTSTR( 0.46523, 2, TRUE );
```

The result is: '0.47';

Due to the internal representation of floating point numbers and rounding errors during conversion, the resulting string may not exactly match the value of the *in* variable. The maximum number of significant digits for LREAL variables is limited to 15.

Sample:

```
PROGRAM MAIN
VAR
    double  : LREAL;
    s1      : STRING;
    s2      : STRING;
    s3      : STRING;
    s4      : STRING;
END_VAR
```

```
double := 0.5;
s1 := LREAL_TO_FMTSTR( double, 25,  FALSE );
s2 := LREAL_TO_FMTSTR( double, 2,  FALSE );
s3 := LREAL_TO_FMTSTR( double, 0,  TRUE );
s4 := LREAL_TO_FMTSTR( double, 2,  TRUE );
```

The result is:

s1 = '0.4999999999999999756000000' This is how *double* variables are represented internally. This number is used as the starting point for the rounding operation.

s2 = '0.49'

Rounding leads to the following results:

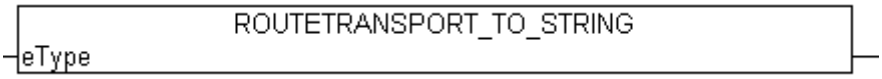
s3 = '0'

s4 = '0.50'

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.40 ROUTETRANSPORT_TO_STRING



The function converts the AMS Message Router transport layer identifier into a string.

FUNCTION ROUTETRANSPORT_TO_STRING : STRING

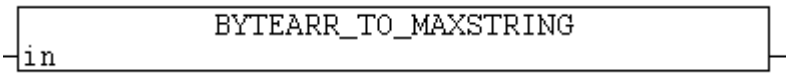
```
VAR_INPUT
    eType   : E_RouteTransportType;
END_VAR
```

eType:: Transport layer identifier [▶ 236] to be converted.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.41 BYTEARR_TO_MAXSTRING



Converts byte array ASCII codes to string.

FUNCTION BYTEARR_TO_MAXSTRING : T_MaxString

T_MaxString

VAR_INPUT

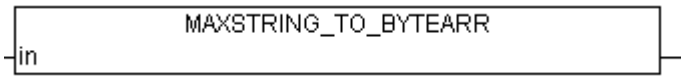
```
VAR_INPUT
    in : ARRAY[0..MAX_STRING_LENGTH] OF BYTE;
END_VAR
```

in: Byte array variable (MAX_STRING_LENGTH default value: 255).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86, ARM)	TcUtilities.Lib

3.42 MAXSTRING_TO_BYTEARR



Convertes a string in single ASCII codes of a byte array.

FUNCTION MAXSTRING_TO_BYTEARR: ARRAY[0..MAX_STRING_LENGTH] OF BYTE

VAR_INPUT

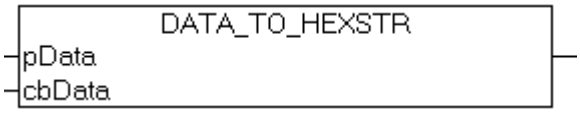
```
VAR_INPUT
    in : T_MaxString;
END_VAR
```

in: String to be converted (T_MaxString) .

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.43 DATA_TO_HEXSTR



The function converts binary data into a hexadecimal string. This function can be used for converting simple data types and structure variables. The maximum length of the binary data must not exceed 85 bytes. If the maximum length is exceeded, a dot ('.') is added to the results string and the conversion is aborted. The remaining data bytes are not converted. In the event of faulty function parameters (*pData* = zero or *cbData* = zero) the function returns an empty string.

FUNCTION DATA_TO_HEXSTR : T_MaxString

T_MaxString

```
VAR_INPUT
    pData   : DWORD;
    cbData  : UDINT(0..85);
    bLoCase : BOOL := FALSE;
END_VAR
```

pData: Start address (pointer) for the binary data to be converted. The address can be determined with the ADR operator.

cbData: Max. length of the binary data to be converted. The length may not exceed 85 bytes. The length can be determined with the SIZEOF operator.

bLoCase: This parameter determines whether capital or lower-case letters are to be used in the conversion. TRUE = lower-case letters, FALSE = capital letters.

Example in ST:

Please ensure that that the data size of the *overflow* variables does not exceed 85 bytes. For this reason, a dot is added to the results string *sH5*.

The byte order in the *number* variable is interchanged, since the memory organization of the counter variable is based on the little-endian-format (also referred to as Intel format).

```
PROGRAM MAIN
VAR
    str     : T_MaxString := 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
```

```
    number   : DWORD := 16#BECF1234;
    char     : BYTE := 16#07;
    null     : UDINT := 0;

    overflow : ARRAY[0..86] OF BYTE; (* data overflow *)
    cbOverflow : UDINT;

    sH1, sH2, sH3, sH4, sH5 : T_MaxString;
END_VAR
```

```
sH1 := DATA_TO_HEXSTR( pData := ADR(str), cbData := LEN(str), FALSE );
sH2 := DATA_TO_HEXSTR( pData := ADR(number), cbData := SIZEOF(number), FALSE );
sH3 := DATA_TO_HEXSTR( pData := ADR(char), cbData := SIZEOF(char), FALSE );
sH4 := DATA_TO_HEXSTR( pData := ADR(null), cbData := SIZEOF(null), FALSE );
cbOverflow:= SIZEOF(overflow);
sH5 := DATA_TO_HEXSTR( pData := ADR(overflow), cbData := cbOverflow, FALSE );
```

The result:

sH1 = '61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 30 31 32 33 34 35 36 37 38 39'

sH2 = '34 12 CF BE'

sH3 = '07'

sH4 = '00 00 00 00'

sH5 = '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.'

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1331	PC or CX (x86, ARM)	TcUtilities.Lib

3.44 HEXSTR_TO_DATA

HEXSTR_TO_DATA
sHex
pData
cbData

The function converts a hexadecimal string into binary data and returns the number of successfully converted data bytes as result. Only spaces may be used as separators in the hexadecimal string to be converted. Lower and upper case letters are permitted as hex characters. In the event of an error or an illegal character the conversion is aborted and a zero length is returned as result.

FUNCTION HEXSTR_TO_DATA : UDINT

```
VAR_INPUT
    sHex     : T_MaxString;
    pData    : DWORD;
    cbData   : UDINT;
END_VAR
```

sHex: The hexadecimal string to be converted. E.g.: 'AB CD 01 23'

pData: Start address (pointer) to the target buffer into which the converted data bytes are to be written. The address can be determined with the ADR operator.

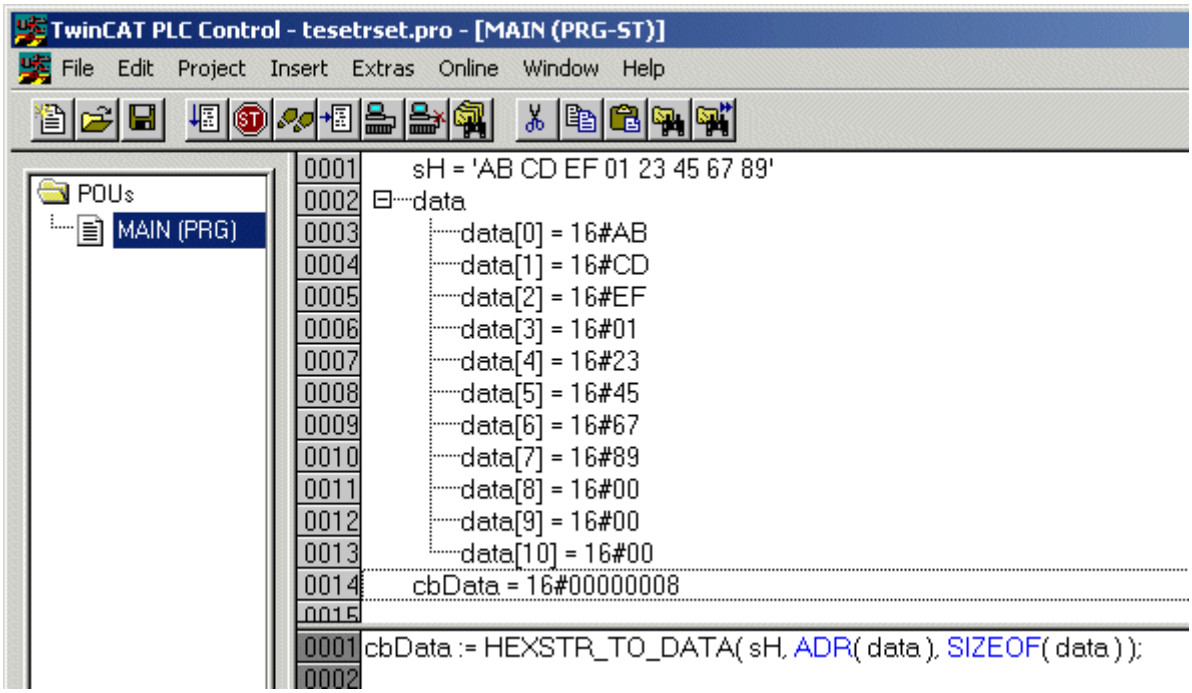
cbData: Max. available length of the target buffer. The length can be determined with the SIZEOF operator.

Example in ST:

```
PROGRAM MAIN
VAR
    sH      : STRING := 'AB CD EF 01 23 45 67 89';
    data    : ARRAY[0..10] OF BYTE;
    cbData  : UDINT;
END_VAR
```

```
cbData := HEXSTR_TO_DATA( sH, ADR( data ), SIZEOF( data ) );
```

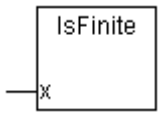
The result (online):



Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1334	PC or CX (x86, ARM)	TcUtilities.Lib

3.45 IsFinite



The function IsFinite() returns TRUE, if its argument has a finite value (INF < x < +INF). The function returns FALSE, if the argument is infinite or NaN (NaN = Not a number). IsFinite() checks whether the formatting of an LREAL or REAL variable complies with IEEE.

INF numbers may occur in a runtime system if the result of a mathematical operation falls outside the range that can be represented. E.g.:

```
PROGRAM MAIN
VAR
    fSingle : REAL := 12.34;
END_VAR
```

```
(*Cyclic called program code*)
fSingle := fSingle*2;
```

NaN numbers may occur in the runtime system if their actual formatting (memory content) was overwritten through illegal access (e.g. by using the MEMCPY of MEMSET functions). E.g.:

```
PROGRAM MAIN
VAR
    fSingle : REAL := 12.34;
END_VAR
```

```
(*Cyclic called program code*)
MEMSET( ADR( fSingle ), 16#FF, SIZEOF( fSingle ) ); (* Invalid initialization of REAL variable *)
```

Calling a conversion function with an NaN or INF number as parameter causes an FPU exception on a PC system (i368). This exception subsequently leads to the PLC being stopped. The function IsFinite() enables the value of the variables to be checked, and therefore the FPU exception to be avoided and program execution to be continued.

FUNCTION IsFinite : BOOL

```
VAR_INPUT
    x :T_Arg;
END_VAR
```

x: an auxiliary structure [▶ 239] with information about the REAL or LREAL variables to be checked. The structure parameters have to be generated when IsFinite() is called from helper functions F_REAL [▶ 104] or F_LREAL [▶ 104] and transferred as parameters.

Sample of a call in ST:

In the following sample, the formatting of a REAL and an LREAL variable is checked, and an FPU exception is avoided.

```
PROGRAM MAIN
VAR
    fSingle         : REAL := 12.34;
    fDouble         : LREAL := 56.78;
    singleAsString  : STRING;
    doubleAsString  : STRING;
END_VAR
```

```
fSingle := fSingle*2;
IF IsFinite( F_REAL( fSingle ) ) THEN
    singleAsString := REAL_TO_STRING( fSingle );
ELSE
  (* report error !*)
 fSingle := 12.34;
END_IF

fDouble := fDouble*2;
IF IsFinite( F_LREAL( fDouble ) ) THEN
    doubleAsString := LREAL_TO_STRING( fDouble );
ELSE
  (* report error !*)
 fDouble := 56.78;
END_IF
```

In the following case, an FPU exception cannot be avoided through checking with IsFinite():

```
PROGRAM MAIN
VAR
    bigFloat  : LREAL := 3.0E100;
    smallDigit: INT;
END_VAR
```

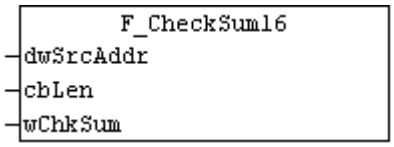
```
IF IsFinite( F_LREAL( bigFloat ) ) THEN
    smallDigit := LREAL_TO_INT( bigFloat );
END_IF
```

While the bigFloat variable has the right formatting, the variable value is too large for conversion into an INT type. An exception is triggered on a PC system (i368), and the runtime system is stopped.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.46 F_CheckSum16



The function "F_CheckSum16" performs a 16 bit check sum of any data.

FUNCTION F_CheckSum16 : WORD

```
VAR_INPUT
    dwSrcAddr    : DWORD;
    cbLen        : UDINT;
    wChkSum      : WORD;
END_VAR
```

dwSrcAddr: Address of data buffer.

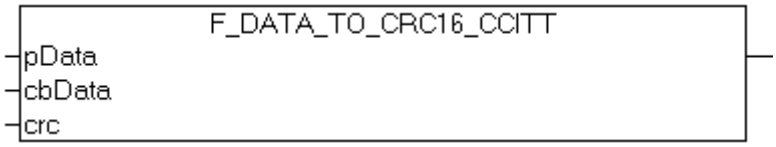
cbLen: Length of data buffer.

wChkSum: Initial value = 0 or last check sum.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 735	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.47 F_DATA_TO_CRC16_CCITT



The function "F_DATA_TO_CRC16_CCITT" performs a 16-Bit-CRC-CCITT (cyclic redundancy check) of any data. Internaly the function F_BYTE_TO_CRC16_CCITT [▶ 54] is used, refer to the documentation of the F_BYTE_TO_CRC16_CCITT [▶ 54] function for further information.

FUNCTION F_DATA_TO_CRC16_CCITT: WORD

```
VAR_INPUT
    pData   : DWORD;(* Pointer to data *)
    cbData  : UDINT;(* Length of data *)
    crc     : WORD; (* Initial value (16#FFFF or 16#0000) or previous CRC-16 result *)
END_VAR
```

pData: Address of data buffer.

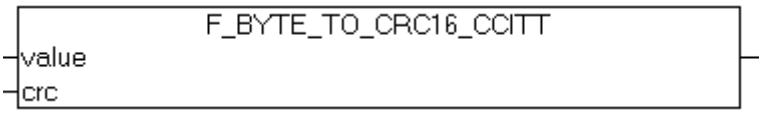
cbData: Length of data buffer.

crc: Initial value = 16#FFFF or 16#0000 or last CRC.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1524	PC or CX (x86, ARM)	TcUtilities.Lib

3.48 F_BYTE_TO_CRC16_CCITT



The function "F_BYTE_TO_CRC16_CCITT" performs a 16-Bit-CRC-CCITT (cyclic redundancy check) of single data bytes.

Used CRC-16 CCITT generator polynomial:

- Name : CRC-16 CCITT
- Standards : CRC-CCITT
- References : ITU X.25/T.30, ADCCP, SDLC/HDLC, ...
- Polynomial value : 0x1021
- Polynom : x^16 + x^12 + x^5 + 1

FUNCTION F_BYTE_TO_CRC16_CCITT : WORD

```
VAR_INPUT
    value   : BYTE;(* Data value *)
    crc     : WORD;(* Initial value (16#FFFF or 16#0000) or previous CRC-16 result *)
END_VAR
```

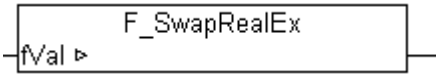
value: The input data byte.

crc: Initial value = 16#FFFF or 16#0000 or last CRC.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1524	PC or CX (x86, ARM)	TcUtilities.Lib

3.49 F_SwapRealEx



The way in which a REAL number is represented in the memory of a bus controller (165) is different from the way a REAL number is represented in the memory of an Intel system (PC).
In order to represent a bus controller's REAL number correctly in a PC, it is necessary for the high and low words of the REAL number to be swapped. Under the programming environment this is already done in online or simulation mode. If it is desired to request the REAL data from a bus controller over the network (ADS protocol, ADSDLL, AdsOcx etc.) and to display it correctly in an Intel PC, then it is necessary to convert the REAL data into the correct format. This may take place on the bus controller side or the PC side.

The function F_SwapReal can be used to convert the REAL variables (e.g. variables to be read by a VB application or recorded with TwinCAT Scope View) into a suitable format on the PC side. The function changes the memory representation of the transferred *fVal* parameter (VAR_IN_OUT).

FUNCTION F_SwapRealEx : BOOL

VAR_IN_OUT

```
VAR_IN_OUT
    fVal    :REAL;
END_VAR
```

fVal: The REAL value to be converted.

Return parameter	Meaning
TRUE	No error.
FALSE	Error during function execution

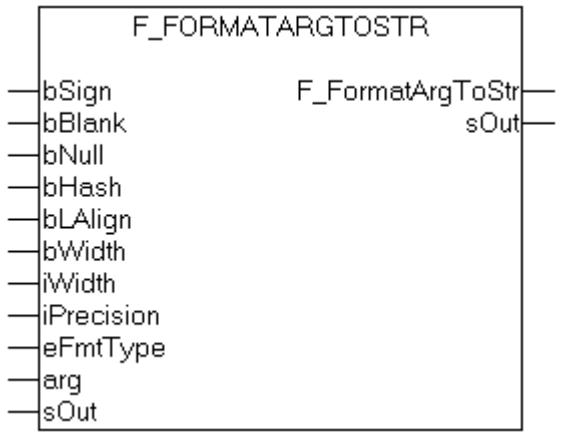
Example:

See: Example: Communication BC/BX<->PC/CX (F_SwapRealEx). [▶ 252]

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10 Build > 1301	PC or CX (x86) CX (ARM)	TcUtilities.Lib

3.50 F_FormatArgToStr



Auxiliary format function. This function is used internally by the FB_FormatString [▶ 157] function block. The function can be used to convert a variable of type T_Arg [▶ 239] into a formatted string according to the format specification [▶ 275].

FUNCTION F_FormatArgToStr : UDINT

```
VAR_INPUT
    bSign       : BOOL;         (* Sign prefix flag *)
    bBlank      : BOOL;         (* Blank prefix flag *)
    bNull       : BOOL;         (* Null prefix flag *)
    bHash       : BOOL;         (* Hash prefix flag *)
    bLAlign     : BOOL;         (* FALSE => Right align (default), TRUE => Left align *)
    bWidth      : BOOL;         (* FALSE => no width padding, TRUE => blank or zeros padding enabled
 *)
    iWidth      : INT;          (* Width length parameter *)
    iPrecision  : INT;          (* Precision length parameter *)
    eFmtType    : E_TypeFieldParam;    (* Format type field parameter *)
    arg         : T_Arg;              (* Format argument *)
END_VAR
VAR_IN_OUT
    sOut        : T_MaxString;
END_VAR
```

bSign: sign flag [▶ 276].

bBlank: blank flag [▶ 276].

bNull: null flag [▶ 276].

bHash: hash prefix flag [▶ 276].

bLAlign: alignment flag [▶ 276] (TRUE=left align).

bWidth: If TRUE, the iWidth parameter is interpreted, otherwise not.

iWidth: Width [▶ 277] parameter.

iPrecision: Precision [▶ 277] parameter.

eFmtType: Type [▶ 275] parameter.

arg: The argument to be formatted. The following auxiliary functions can be used for converting different types of PLC variables into the required data type T_Arg [▶ 239]: F_BYTE, F_WORD, F_DWORD, F_SINT, F_INT, F_DINT, F_USINT, F_UINT, F_UDINT, F_STRING, F_REAL, F_LREAL.

sOut: If successful, this variable returns the formatted output string.

Return parameter	Meaning
0	No error.
<> 0	Error. For error description see Format error codes [▶ 250]

Examples in ST:

Formatting a BYTE variable as a binary string.

```
PROGRAM MAIN
VAR
    s1      : T_MaxString;
    s2      : T_MaxString;
    s3      : T_MaxString;
    s4      : T_MaxString;
    s5      : T_MaxString;
    errID   : UDINT;
    varByte : BYTE;
    double  : LREAL;
    L1      : INT;
    L2      : INT;
    L3      : INT;
```

```
    L4      : INT;
    L5      : INT;
END_VAR
```

```
varByte := 128;
errID := F_FormatArgToStr(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, 20, 8, TYPEFIELD_B, F_BYTE( varB
yte ), s1 );
errID := F_FormatArgToStr(FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, 20, 8, TYPEFIELD_B, F_BYTE( varBy
te ), s2 );
errID := F_FormatArgToStr(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, 20, 8, TYPEFIELD_B, F_BYTE( varByt
e ), s3 );
errID := F_FormatArgToStr(FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, 20, 8, TYPEFIELD_B, F_BYTE( varByte
 ), s4 );
L1 := LEN( s1 );
L2 := LEN( s2 );
L3 := LEN( s3 );
L4 := LEN( s4 );
```

The result:

s1 = '10000000'

s2 = ' 10000000'

s3 = '10000000 '

s4 = '2#10000000 '

L1 = 8

L2 = 20

L3 = 20

L4 = 20

Formatting an LREAL variable.

```
double := 12345.6789;
errID := F_FormatArgToStr( FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, 20, 8, TYPEFIELD_F, F_LREAL( do
uble ), s1 );
errID := F_FormatArgToStr( FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, 20, 8, TYPEFIELD_F, F_LREAL( dou
ble ), s2 );
errID := F_FormatArgToStr( FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, 20, 8, TYPEFIELD_F, F_LREAL( doub
le ), s3 );
errID := F_FormatArgToStr( FALSE, FALSE, TRUE, FALSE, FALSE, TRUE, 20, 8, TYPEFIELD_F, F_LREAL( doub
le ), s4 );
errID := F_FormatArgToStr( TRUE, FALSE, FALSE, TRUE, TRUE, TRUE, 20, 8, TYPEFIELD_F, F_LREAL( double
 ), s5 );
L1 := LEN( s1 );
L2 := LEN( s2 );
L3 := LEN( s3 );
L4 := LEN( s4 );
L5 := LEN( s5 );
```

The result:

s1 = '12345.67890000'

s2 = ' 12345.67890000'

s3 = '12345.67890000 '

s4 = '00000012345.67890000'

s5 = '+12345.67890000 '

L1 = 14

L2 = 20

L3 = 20

L4 = 20

L5 = 20

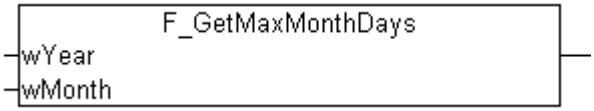
Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Also see about this

- E_TypeFieldParam [▶ 235]
- F_BYTE [▶ 105]
- F_LREAL [▶ 104]

3.51 F_GetMaxMonthDays



F_GetMaxMonthDays returns the number of days in a month.

FUNCTION F_GetMaxMonthDays : WORD

VAR_INPUT

```
VAR_INPUT
    wYear  : WORD;
    wMonth : WORD;
END_VAR
```

wYear: Year.

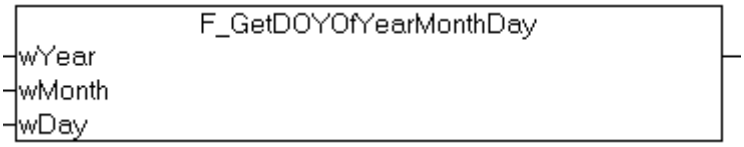
wMonth: Month (1 bis 12).

Return value	Description
0	Invalid wMonth parameter.
> 0	No error. Number of days in a month.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.52 F_GetDOYOfYearMonthDay



The function returns the day number of the year.

FUNCTION F_GetDOYOfYearMonthDay: WORD

VAR_INPUT

```
VAR_INPUT
    wYear   : WORD;(* Year: 0..2xxx *)
    wMonth  : WORD;(* Month 1..12 *)
    wDay    : WORD;(* Day: 1..31 *)
END_VAR
```

wYear: Year (0 ~ 2999).

wMonth: Month (1 ~ 12).

wDay: Day (1 ~ 31).

Return value	Description
0	Invalid wYear, wMonth or wDay parameter.
> 0	No error. Day number (1 ~ 366)

Example:

```
PROGRAM P_TEST_DOY
VAR
    wYear   : WORD;
    wDOY    : WORD;
    wMonth  : WORD;
    wDay    : WORD;
END_VAR
```

```
wYear := 2009;
wMonth := 1;
wDay := 31;
wDOY := F_GetDOYOfYearMonthDay( wYear, wMonth, wDay );(* wDOY = 31 *)

wYear := 2009;
wMonth := 2;
wDay := 1;
wDOY := F_GetDOYOfYearMonthDay( wYear, wMonth, wDay );(* wDOY = 32 *)

wYear := 2009;
wMonth := 3;
wDay := 1;
wDOY := F_GetDOYOfYearMonthDay( wYear, wMonth, wDay );(* wDOY = 60 *)
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1530	PC or CX (x86, ARM)	TcUtilities.Lib

3.53 F_GetMonthOfDOY

The function calculates the month from day number of the year.

FUNCTION F_GetMonthOfDOY: WORD

VAR_INPUT

```
VAR_INPUT
    wYear : WORD;(* Year: 0..2xxx *)
 wDOY : WORD;(* Year's day number: 1..366 *)
END_VAR
```

wYear: Year (0 ~ 2999).

wDOY: Day number of the year (1 ~ 366).

Return value	Description
0	Invalid wYear or wDOY parameter.
> 0	No error. Month (1 ~ 12)

Example:

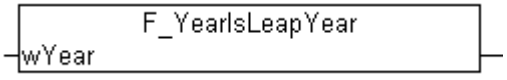
```
PROGRAM P_TEST_DOY
VAR
    wYear       : WORD;
    wDOY        : WORD;
    wMonth      : WORD;
END_VAR
```

```
wYear := 2009;
wDOY := 31;
wMonth:= F_GetMonthOfDOY( wYear, wDOY );(* wMonth = 1 *)

wYear := 2009;
wDOY := 32;
wMonth := F_GetMonthOfDOY( wYear, wDOY );(* wMonth = 2 *)

wYear := 2009;
wDOY := 60;
wMonth := F_GetMonthOfDOY( wYear, wDOY );(* wMonth = 3 *)
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1530	PC or CX (x86, ARM)	TcUtilities.Lib

3.54 F_YearIsLeapYear



F_YearIsLeapYear determines whether the given year is a leap year.

FUNCTION F_YearIsLeapYear: BOOL

VAR_INPUT

```
VAR_INPUT
    wYear : WORD;
END_VAR
```

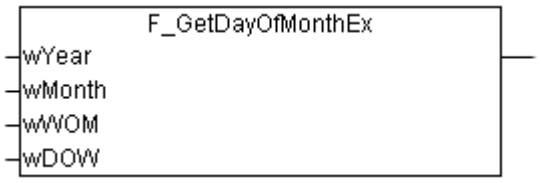
wYear: Year.

Return value	Description
TRUE	wYear is leap year
FALSE	Not leap year

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.55 F_GetDayOfMonthEx



The function calculates the date of the first, second etc. weekday in a particular month and year (e.g. the date of the second Monday in January 2011).

FUNCTION F_GetDayOfMonthEx: WORD

VAR_INPUT

```
VAR_INPUT
    wYear  : WORD(1601..30827);
    wMonth : WORD(1..12);
    wWOM   : WORD(1..5);
    wDOW   : WORD(0..6);
END_VAR
```

wYear: year (1601 to 30827).

wMonth: month (1 to 12).

wWOM: week in month (1 of 5). The value 1 corresponds to week 1, 2 to week 2 and 5 to the last week (even if the month does not have 5 weeks).

wDOW: day of the week (0 to 6). 0 = Sunday, 1 = Monday... 6 = Saturday.

Return parameter	Description
0	Error, wrong or invalid function parameter
> 0	No error. Day of the month

Example:

The example determines the date of the second Monday in August 2011. The result is: 8.

```
PROGRAM P_Dok_F_GetDayOfMonthEx
VAR
    wYear  : WORD := 2011;
    wMonth : WORD := 8;
    wWOM   : WORD(1..5) := 2;(* Week of month: 2 = Second week *)
    wDOW   : WORD(0..6) := 1;(* Day of week 1 = Monday *)
    wDay   : WORD;(* Day of month *)
END_VAR
```

```
wDay := F_GetDayOfMonthEx( wYear, wMonth, wWOM, wDOW );
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

3.56 F_TranslateFileTimeBias

This function converts the input time to the time in another time zone based on the specified bias time shift. The function can be used to convert the local time to UTC time (Universal Time Coordinates) and vice versa, for example.

FUNCTION F_TranslateFileTimeBias: T_FILETIME

T_FILETIME [▶ 239]

VAR_INPUT

```
VAR_INPUT
    in      : T_FILETIME;(* Input time in file time format to be translated *)
    bias    : DINT;
(* Bias value in minutes. The bias is the difference, in minutes, between Coordinated Universal Time
 (UTC) and local time. *)
    toUTC   : BOOL;
(* TRUE => Translate from local time to UTC, FALSE => Translate from UTC to local Time *)
END_VAR
```

in: input time [▶ 239] that is to be converted.

bias: difference between UTC time and local time in minutes (positive or negative values are permitted).

toUTC: this parameter can be used to specify the direction in which the input time is to be converted.

toUTC	Direction	Internal formula
FALSE	UTC -> local time	Local time := UTC - bias
TRUE	Local time -> UTC	UTC := local time + bias

Sample:

The *in* variable contains the time to be converted. The *bToUTC* variable determines the conversion direction. If *bToUTC* = TRUE, the local time is converted to UTC time, if *bToUTC* = FALSE, the UTC time is converted to local time. The *WEST_EUROPE_TZI* constant contains the time zone information for Western Europe. The required bias value is calculated from the time zone information in the constant and the current bDST setting (Daylight Saving Time). Alternatively, the current time zone information of a TwinCAT system can be determined with the function block: FB_GetTimeZoneInformation [▶ 177].

Data type DT was selected for the input time because of the visual control option in online mode. Conversions to other time formats are not necessarily recommend since the conversion functions can be very computing-intensive.

```
PROGRAM MAIN
VAR
    bDST    : BOOL := TRUE;(* TRUE => Daylight saving time, FALSE => Standard time *)
    bToUTC  : BOOL := FALSE;
(* TRUE => Convert local time to UTC time, FALSE => Convert UTC time to local time *)
    in      : DT := DT#2011-08-29-15:15:31;
```

```
    out     : DT;
    bias    : DINT;
END_VAR
```

```
IF bDST THEN
    bias := WEST_EUROPE_TZI.bias + WEST_EUROPE_TZI.daylightBias;
ELSE
    bias := WEST_EUROPE_TZI.bias + WEST_EUROPE_TZI.standardBias;
END_IF

out := FILETIME_TO_DT( F_TranslateFileTimeBias( DT_TO_FILETIME( in ), bias, bToUTC ) );
```

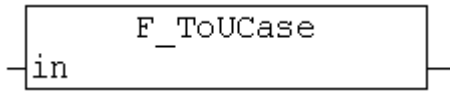
Further time and time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias, FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

3.57 F_ToUCase



The F_ToUCase function converts a specified string to uppercase.

Limited character set available

Per default the conversion function uses the character set of the Windows Code Page 1252 Latin 1, SBCS (Single Byte Character Set). Another character set (but only Windows Code Page 1250 Central European) can be chosen via the global variable GLOBAL_SBCS_TABLE (see example).

FUNCTION F_ToUCase : T_MaxString

T_MaxString

VAR_INPUT

```
VAR_INPUT
    in      : T_MaxString;
END_VAR
```

in: String to be converted.

Example:

```
PROGRAM MAIN
VAR
    sUCase : STRING;
END_VAR
```

```
sUCase := F_ToUCase( 'to upper case 1234567890 äöüß' );
```

The result is: 'TO UPPER CASE 1234567890 ÄÖÜß'

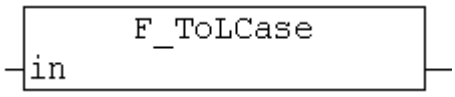
```
GLOBAL_SBCS_TABLE := eSBCS_CentralEuropean;
sUCase := F_ToUCase( 'to upper case 1234567890 ąęśćżźłó' );
```

The result is: 'TO UPPER CASE 1234567890 ĄĘŚĆŻŹŁÓ'

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.58 F_ToLCase



The F_ToLCase function converts a specified string to lowercase.

Limited character set available

Per default the conversion function uses the character set of the Windows Code Page 1252 Latin 1, SBCS (Single Byte Character Set). Another character set (but yet only Windows Code Page 1250 Central European) can be chosen via the global variable GLOBAL_SBCS_TABLE (see example).

FUNCTION F_ToLCase : T_MaxString

T_MaxString

VAR_INPUT

```
VAR_INPUT
    in      : T_MaxString;
END_VAR
```

in: String to be converted.

Example:

```
PROGRAM MAIN
VAR
    sLCase : STRING;
END_VAR
```

```
sLCase := F_ToLCase( 'TO LOWER CASE 1234567890 ÄÖÜß' );
```

The result is: 'to lower case 1234567890 äöüß'

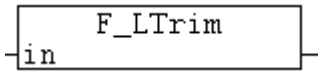
```
GLOBAL_SBCS_TABLE := eSBCS_CentralEuropean;
sLCase := F_ToLCase( 'TO LOWER CASE 1234567890 ĄĘŚĆŻŹŁÓ' );
```

The result is: 'to lower case 1234567890 ąęśćżźłó'

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.59 F_LTrim



The F_LTrim function removes spaces on the left side of a string.

FUNCTION F_LTrim : T_MaxString

T_MaxString

VAR_INPUT

```
VAR_INPUT
    in      :T_MaxString;
END_VAR
```

Example:

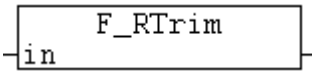
```
PROGRAM MAIN
VAR
    sLTrim : STRING;
END_VAR
```

```
sLTrim := F_LTrim(' <trim '); (* result: '<trim ' *)
sLTrim := F_LTrim(' <trim'); (* result: '<trim' *)
sLTrim := F_LTrim('<trim'); (* result: '<trim' *)
sLTrim := F_LTrim(''); (* result: '' *)
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.60 F_RTrim



The F_RTrim function removes spaces on the right side of a string.

FUNCTION F_RTrim : T_MaxString

T_MaxString

VAR_INPUT

```
VAR_INPUT
    in      :T_MaxString;
END_VAR
```

Example:

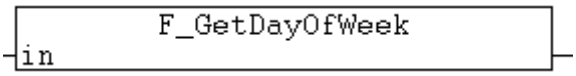
```
PROGRAM MAIN
VAR
    sRTrim  : STRING;
    sLRTrim : STRING;
END_VAR
```

```
sRTrim := F_RTrim(' trim> '); (* result: ' trim>' *)
sRTrim := F_RTrim('trim> '); (* result: 'trim>' *)
sRTrim := F_RTrim('trim>'); (* result: 'trim>' *)
sRTrim := F_RTrim(''); (* result: '' *)

sLRTrim := F_RTrim( F_LTrim( ' <trim> ')); (* result: '<trim>' *)
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.61 F_GetDayOfWeek



You can use this function to obtain the day of week represented by a specified date and time value. The function uses the ISO 8601 standard definition of a week. A week is considered to start a Monday = 1and end on Sunday = 7.

FUNCTION F_GetDayOfWeek : WORD

VAR_INPUT

```
VAR_INPUT
    in : DT;
END_VAR
```

in: Specified date and time value.

Example in structured text:

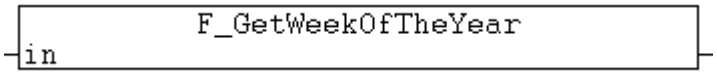
```
PROGRAM MAIN
VAR
    dtFirst : DT := DT#2008-01-01-00:00;
    dayOfWeek : WORD;
END_VAR
dayOfWeek := F_GetDayOfWeek(dtFirst);
```

The result is: 2 (Tuesday)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1325	PC or CX (x86, ARM)	TcUtilities.Lib

3.62 F_GetWeekOfTheYear



You can use this function to obtain the week of the year represented by a specified date and time value. The function uses the ISO 8601 standard definition of a week.

- The first week of a year is defined as the **first week with four or more days in that year (DIN 1355 / ISO 8601)**;
- A week is considered to start an a Monday and end on Sunday.
- The returned value is between 1 and 53. The first week of a year has the number 1;
- If the first calendar day of the year is a Friday, Saturday, or Sunday, then for the first three, two, or one days of the calendar year, the F_GetWeekOfTheYear returns the last week of the previous year;
- If the last calendar day of the year is a Monday, Tuesday, or Wednesday, then for the last one, two, or three days of the calendar year, F_WeekOfTheYear returns 1 (the first week of the next calendar year);

FUNCTION F_GetWeekOfTheYear : WORD

VAR_INPUT

```
VAR_INPUT
    in : DT;
END_VAR
```

in: Specified date and time value.

Example in structured text:

```
PROGRAM MAIN
VAR
    dtNow      : DT := DT#2008-03-17-12:00;
    weekOfYear : WORD;
END_VAR
```

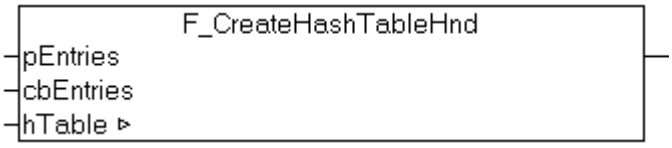
```
weekOfYear := F_GetWeekOfTheYear(dtNow);
```

The result is: 12.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1325	PC or CX (x86, ARM	TcUtilities.Lib

3.63 F_CreateHashTableHnd



The function initialises the hash table handle. The table handle must be initialised once by calling the F_CreateHashTableHnd function.

FUNCTION F_CreateHashTableHnd : BOOL

VAR_INPUT

```
VAR_INPUT
    pEntries  : POINTER TO T_HashTableEntry := 0; (* Pointer to the first entry of hash table databa
se (element array) *)
    cbEntries : UDINT := 0; (* Byte size (length) of hash table database (element array) *)
END_VAR
```

pEntries: Address of the first T_HashTableEntry [▶ 242] array element. The address can be determined with the ADR operator.

cbEntries: T_HashTableEntry byte size. The byte size can be determined with the SIZEOF operator.

VAR_IN_OUT

```
VAR_IN_OUT
    hTable : T_HHASHTABLE;(* Hash table handle *)
END_VAR
```

hTable: Hash table handle [▶ 242] to be initialized. The handle is required for accessing the hash table from the function block FB_HashTableCtrl [▶ 190].

Return parameter	Description
TRUE	No error
FALSE	Error

Example: Please refer to the documentation for the FB_HashTableCtrl [▶ 190] function block.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

3.64 F_CreateLinkedListHnd

The function initialises the Linked List Handle. The List Handle must be initialised once by calling the *F_CreateLinkedListHnd* function.

FUNCTION F_CreateLinkedListHnd : BOOL

VAR_INPUT

```
VAR_INPUT
    pEntries : POINTER TO T_LinkedListEntry := 0; (* Pointer to the first linked list node (element
array) *)
    cbEntries : UDINT := 0; (* Byte size (length) of linked list node array *)
END_VAR
```

pEntries: The address of the first T_LinkedListEntry [▶ 243] array element. The address can be determined with the ADR operator.

cbEntries: The size of the *T_LinkedListEntry* array in bytes. The byte size can be determined with the SIZEOF operator.

VAR_IN_OUT

```
VAR_IN_OUT
    hList : T_HLINKEDLIST;(* Linked list handle *)
END_VAR
```

hList: Hash table handle [▶ 242] to be initialized. The handle is required for accessing the list from the function block FB_LinkedListCtrl [▶ 192].

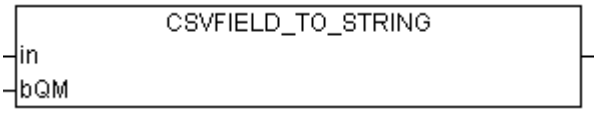
Return parameter	Description
TRUE	No error
FALSE	Error

Example: See the documentation for the FB_LinkedListCtrl [▶ 192] function block.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcUtilities.Lib

3.65 CSVFIELD_TO_STRING



The function converts a data field in CSV data field format that is present as a source string into a value in PLC string format. Double quotation marks in the data field are replaced with simple quotation marks. If the bQM parameter is set (QM = quotation marks) the outer quotation marks (around the data field) are removed from the source string. If successful the function returns the convert string as the result. The function returns an empty string if an error occurred during the conversion, but only if the source string was not an empty string.

The function is usually used together with the function block FB_CSVMemBufferReader [▶ 227] in order to read (interpret) data sets that are stored in the PLC memory in CSV format. Before this operation the CSV data sets are usually read from a file into the PLC memory. The source string must not contain binary data. Binary data with the value zero would terminate and truncate the string in the wrong place. To convert data fields with binary data please use the function CSVFIELD_TO_ARG [▶ 70].

FUNCTION CSFIELD_TO_STRING : T_MaxString

T_MaxString

```
VAR_INPUT
    in      : T_MaxString;
    bQM     : BOOL;
END_VAR
```

in: Source string with a data field in CSV format that is to be converted into a value in PLC string format.

bQM: If this input is TRUE the enclosing quotation marks are removed from the source string.

bQM	Description	Source string	Result string	CSV-compliant
FALSE	Source string without enclosing quotation marks should only contain letters and numbers. In this case the source string must not contain any non-printable control character, quotation marks, semicolons, commas (US CSV format) or binary data.			
		'Module_XA5'	'Module_XA5'	Yes
		'123456'	'123456'	Yes
		''	''	Yes
		'A""""B'	'A""B'	Nein
		'A""B'	'A"B'	No
		';'	';'	No
		'RN'	'RN'	No
		'AB$00CD'	'AB' (string was truncated)	No
TRUE	A source string that is not enclosed in quotation marks should not contain any non-printable control character, quotation mark, semicolon, or comma (US CSV format). Binary data are not permitted.	'"Module_XA5"'	'Module_XA5'	Yes
		'"123456"'	'123456'	Yes
		'""'	''	Yes
		'"A""""B"'	'A""B'	Yes
		'"A""B"'	'A"B'	Yes
		'";"'	';'	Yes
		'"RN"'	'RN'	Yes
		'"AB$00CD"'	'AB' (string was truncated)	No

Example in ST:

```
PROGRAM MAIN
VAR
    s1 : STRING;
    s2 : STRING;
END_VAR
```

```
s1 := CSVFIELD_TO_STRING( '"ab_$04_$05_cd-""ALFA""_5"', TRUE );
s2 := CSVFIELD_TO_STRING( 'Module_50', FALSE );
```

The result:

s1 = 'ab_$04_$05_cd-"ALFA"_5'

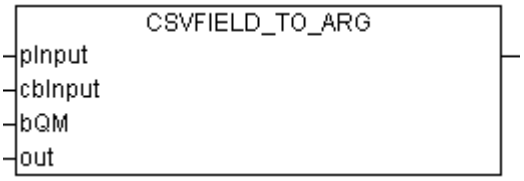
s2 = 'Module_50'

Further information can be found here: Example: Writing/reading of CSV file [▶ 270].

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.66 CSVFIELD_TO_ARG



The function converts the value from a data field in CSV format which is present as byte buffer into a PLC variable. Double quotation marks in the data field are replaced with simple quotation marks. If the bQM parameter is set (QM = quotation marks) the outer quotation marks (around the data field) are removed from the input data. If successful, the function returns the length of the converted data. In the event of an error or if the length of the input data is zero the function returns the value zero. The application must ensure the PLC target variable is large enough to accommodate the value.

The function is usually used together with the function block FB_CSVMemBufferReader [▶ 227] in order to read (parse) data sets that are stored in the PLC memory in CSV format. Before this operation the CSV data sets are usually read from a file into the PLC memory. In contrast to the CSVFIELD_TO_STRING [▶ 69] function this function can also be used to convert CSV data fields with binary data into PLC variables.

FUNCTION CSVFIELD_TO_ARG : UDINT

```
VAR_INPUT
    pInput  : DWORD;
    cbInput : UDINT;
    bQM     : BOOL;
    out     : T_Arg;
END_VAR
```

pInput: Start address (pointer) to a byte buffer containing the data field to be converted in CSV format. The address can be determined with the ADR operator.

cbInput: Length of the data field to be converted in bytes. The length can be determined with the SIZEOF operator.

bQM: If this input is TRUE the enclosing quotation marks are removed from the field data.

out: PLC target variable into which the value of the data field [▶ 239] is to be written.

Example in ST:

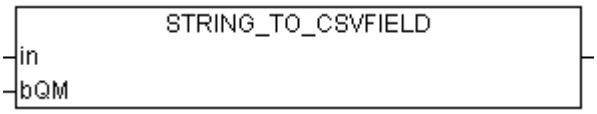
See example in the documentation for the ARG_TO_CSVFIELD [▶ 72] function block.

Further information can be found here: Example: Writing/reading of CSV file [▶ 270].

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.67 STRING_TO_CSVFIELD



The function converts the value of a PLC string variable to a string data field in CSV format. Single quotation marks in the source string are replaced with double quotation marks. If the bQM parameter is set (QM = quotation marks) the outer quotation marks (around the CSV data field) are also added. If successful, the function returns the convert string as the result. The function returns an empty string if an error occurred during the conversion, but only if the source string was not an empty string.

The function is usually used together with the function block FB_CSVMemBufferWriter [▶ 228] to generate data sets in the PLC memory in CSV format. In the next step the memory content can be written to the file.

The source string must not contain binary data. Binary data with the value zero would terminate and truncate the string in the wrong place. To convert binary data please use the function ARG_TO_CSVFIELD [▶ 72].

FUNCTION STRING_TO_CSVFIELD : T_MaxString

T_MaxString

```
VAR_INPUT
    in      : T_MaxString;
    bQM     : BOOL;
END_VAR
```

in: Source string whose value is to be converted into a data field in CSV format.

bQM: If this input is TRUE the enclosing quotation marks are added from the result string.

bQM	Description	Source string	Result string	CSV-compliant
FALSE	Source string without enclosing quotation marks should only contain letters and numbers. In this case the source string must not contain any non-printable control character, quotation mark, semicolon, comma (US CSV format) or binary data.			
		'Module_XA5'	'Module_XA5'	Yes
		'123456'	'123456'	Yes
		''	''	Yes
		'A""B'	'A""""B'	No
		'A"B'	'A""B'	No
		';'	';'	No
		'RN'	'RN'	No
		'AB$00CD'	'AB' (string was truncated)	No
TRUE	A source string that is not enclosed in quotation marks should not contain any non-printable control character, quotation mark, semicolon, or comma (US CSV format). Binary data are not permitted.	'Module_XA5'	'"Module_XA5"'	Yes
		'123456'	'"123456"'	Yes
		''	'""'	Yes
		'A""B'	'"A""""B"'	Yes
		'A"B'	'"A""B"'	Yes
		';'	'";"'	Yes
		'RN'	'"RN"'	Yes
		'AB$00CD'	'"AB" (String was truncated)	No

Example in ST:

```
PROGRAM MAIN
VAR
    s1 : STRING;
    s2 : STRING;
END_VAR
```

```
s1 := STRING_TO_CSVFIELD( 'Module_"ALFA_$05"_6', TRUE );
s2 := STRING_TO_CSVFIELD( 'Module_50', FALSE );
```

The result:

s1 = '"Module_""ALFA_$05""_6"'

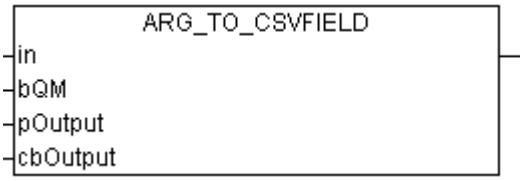
s2 = 'Module_50'

Further information can be found here: Example: Writing/reading of CSV file [▶ 270].

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.68 ARG_TO_CSVFIELD



The function converts the value of a PLC variable to a data field in CSV format. Single quotation marks in the source file are replaced with double quotation marks. If the bQM parameter is set (QM = quotation marks) the outer quotation marks (around the CSV data field) are also added. If successful, the function returns the length of the converted data as the result. The function returns zero if a conversion error occurred or in the event of missing data. The result is written into the byte buffer provided. The application must ensure that the buffer is large enough to contain the result.

The function is usually used together with the function block FB_CSVMemBufferWriter [▶ 228] to generate data sets in the PLC memory in CSV format. In the next step the memory content can be written to the file. In contrast to the STRING_TO_CSVFIELD [▶ 71] function this function can also be used to convert PLC variables with binary data into CSV data fields.

FUNCTION ARG_TO_CSVFIELD : UDINT

```
VAR_INPUT
    in       : T_Arg;
    bQM      : BOOL;
    pOutput  : DWORD;
    cbOutput : UDINT;
END_VAR
```

in: PLC source variable whose value is to be converted into a data field [▶ 239] in CSV format.

bQM: If this input is TRUE the converted field data are enclosed in quotation marks.

pOutput: Start address (pointer) for the output buffer. The buffer address can be determined with the ADR operator. The result data are written into this buffer.

cbOutput: The maximum available size of the output buffer in bytes. The length of the output buffer can be determined with the SIZEOF operator.

Example in ST:

The following example illustrates how PLC variables of different types can be converted to CSV format and vice versa. With ARG_TO_CSVFIELD conversion the result is copied into the byte buffer (field1..field6). With CSVFIELD_TO_ARG [▶ 70] conversion the source data are in the byte buffer (field1..field6), and the result is copied into the TwinCAT PLC variable.

```
PROGRAM P_ArgToConvExample
VAR
    (* PLC data to be converted to or from CSV format *)
    bOperating      : BOOL := TRUE;
    fAxPos      : LREAL := 12.2;
    nCounter        : UDINT := 7;
    sName       : T_MaxString := 'Module: "XAF", $04$05, 20';
    binData         : ARRAY[0..9] OF BYTE := 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
    sShort      : STRING(10) := 'XAF';

    (* conversion buffer *)
    field1 : ARRAY[0..50 ] OF BYTE;
    field2 : ARRAY[0..50 ] OF BYTE;
    field3 : ARRAY[0..50 ] OF BYTE;
    field4 : ARRAY[0..50 ] OF BYTE;
    field5 : ARRAY[0..50 ] OF BYTE;
    field6 : ARRAY[0..50 ] OF BYTE;

    cbField1 : UDINT;
    cbField2 : UDINT;
    cbField3 : UDINT;
    cbField4 : UDINT;
    cbField5 : UDINT;
    cbField6 : UDINT;

    cbVar1 : UDINT;
    cbVar2 : UDINT;
    cbVar3 : UDINT;
    cbVar4 : UDINT;
    cbVar5 : UDINT;
    cbVar6 : UDINT;
END_VAR
```

```
cbField1 := ARG_TO_CSVFIELD( F_BOOL( bOperating ), TRUE, ADR( field1 ), SIZEOF( field1 ) );
cbField2 := ARG_TO_CSVFIELD( F_LREAL( fAxPos ), TRUE, ADR( field2 ), SIZEOF( field2 ) );
cbField3 := ARG_TO_CSVFIELD( F_UDINT( nCounter ), TRUE, ADR( field3 ), SIZEOF( field3 ) );
cbField4 := ARG_TO_CSVFIELD( F_STRING( sName ), TRUE, ADR( field4 ), SIZEOF( field4 ) );
cbField5 := ARG_TO_CSVFIELD( F_BIGTYPE( ADR( binData ), SIZEOF( binData ) ), TRUE, ADR( field5 ), SI
ZEOF( field5 ) );
cbField6 := ARG_TO_CSVFIELD( F_BIGTYPE( ADR( sShort ), LEN( sShort ) ), TRUE, ADR( field6 ), SIZEOF(
 field6 ) );

cbVar1 := CSVFIELD_TO_ARG( ADR( field1 ), cbField1, TRUE, F_BOOL( bOperating ) );
cbVar2 := CSVFIELD_TO_ARG( ADR( field2 ), cbField2, TRUE, F_LREAL( fAxPos ) );
cbVar3 := CSVFIELD_TO_ARG( ADR( field3 ), cbField3, TRUE, F_UDINT( nCounter ) );
cbVar4 := CSVFIELD_TO_ARG( ADR( field4 ), cbField4, TRUE, F_STRING( sName ) );
cbVar5 := CSVFIELD_TO_ARG( ADR( field5 ), cbField5, TRUE, F_BIGTYPE( ADR( binData ), SIZEOF( binData
 ) ) );
cbVar6 := CSVFIELD_TO_ARG( ADR( field6 ), cbField6, TRUE, F_BIGTYPE( ADR( sShort ), LEN( sShort ) )
);
```

The result (byte buffer as hexadecimal string):

cbField1 = 3, field1 = '22 01 22'

cbField2 = 10, field2 = '22 66 66 66 66 66 66 28 40 22'

cbField3 = 6, field3 = '22 07 00 00 00 22'

cbField4 = 25, field4 = '22 4D 6F 64 75 6C 65 3A 20 22 22 58 41 46 22 22 2C 20 04 05 2C 20 32 30 22'

cbField5 = 12, field5 = '22 00 01 02 03 04 05 06 07 08 09 22'

cbField6 = 5, field6 = '22 58 41 46 22'

cbVar1 = 1

cbVar2 = 8

cbVar3 = 4

cbVar4 = 22

cbVar5 = 10

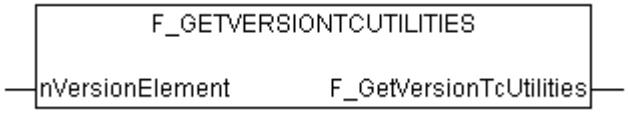
cbVar6 = 3

Further information can be found here: Example: Writing/reading of CSV file. [▶ 270]

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.69 F_GetVersionTcUtilities



The function returns library version info.

FUNCTION F_GetVersionTcUtilities : UINT

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

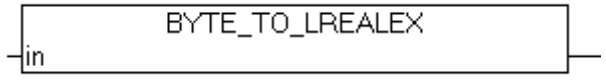
nVersionElement : Version parameter:

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

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.70 BYTE_TO_LREALEX



The conversion of unsigned numbers into floating point numbers of the type LREAL is not supported in TwinCAT 2 on the ARM platform. Unsigned numbers with the highest significant bit set may possibly be implicitly converted into negative floating point numbers. The function described here allows the explicit conversion of the type BYTE into a positive floating point number of the type LREAL in TwinCAT 2 (even if the highest significant bit was set and without compiler warning). Unsigned numbers of the type BYTE are always (implicitly and explicitly) converted into positive floating point numbers in TwinCAT3.

FUNCTION BYTE_TO_LREALEX : LREAL

```
VAR_INPUT
    in      : BYTE;
END_VAR
```

in: Number to be converted.

Sample:

```
PROGRAM MAIN
VAR
    nByte  : BYTE := 16#FF;
    fLreal : LREAL := 0.0;
END_VAR
```

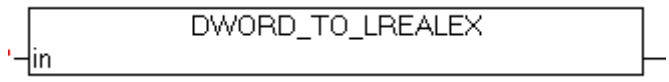
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nByte	+255, Warning 1105*	+255	+255
fLreal := BYTE_TO_LREAL(nByte)	+255, Warning 1105*	+255	+255
fLreal := BYTE#16#FF	+255, Warning 1105*	+255	+255
fLreal := 16#FF	+255	+255	+255
fLreal := BYTE_TO_LREALEX(nByte)	+255	+255	+255
fLreal := BYTE_TO_LREALEX(BYTE#16#FF)	+255	+255	+255
fLreal := BYTE_TO_LREALEX(16#FF)	+255	+255	+255

* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.71 DWORD_TO_LREALEX



The conversion of unsigned numbers into floating point numbers of the type LREAL is not supported in TwinCAT 2 on the ARM platform. Unsigned numbers with the highest significant bit set may possibly be implicitly converted into negative floating point numbers. The function described here allows the explicit conversion of the type DWORD into a positive floating point number of the type LREAL in TwinCAT 2 (even if the highest significant bit was set and without compiler warning). Unsigned numbers of the type DWORD are always (implicitly and explicitly) converted into positive floating point numbers in TwinCAT3.

FUNCTION DWORD_TO_LREALEX : LREAL

```
VAR_INPUT
    in      : DWORD;
END_VAR
```

in: Number to be converted.

Sample:

```
PROGRAM MAIN
VAR
    nDword : DWORD := 16#FFFFFFFF;
    fLreal : LREAL := 0.0;
END_VAR
```

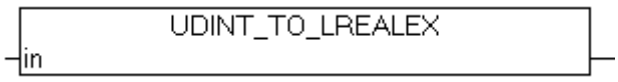
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nDword	-1, Warning 1105*	+4294967295	+4294967295
fLreal := DWORD_TO_LREAL(nDword)	-1, Warning 1105*	+4294967295	+4294967295
fLreal := DWORD#16#FFFFFFFF	-1, Warning 1105*	+4294967295	+4294967295
fLreal := 16#FFFFFFFF	+4294967295	+4294967295	+4294967295
fLreal := DWORD_TO_LREALEX(nDword)	+4294967295	+4294967295	+4294967295
fLreal := DWORD_TO_LREALEX(DWORD#16#FFFFFFFF)	+4294967295	+4294967295	+4294967295
fLreal := DWORD_TO_LREALEX(16#FFFFFFFF)	+4294967295	+4294967295	+4294967295

* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.72 UDINT_TO_LREALEX



The conversion of unsigned numbers into floating point numbers of the type LREAL is not supported in TwinCAT 2 on the ARM platform. Unsigned numbers with the highest significant bit set may possibly be implicitly converted into negative floating point numbers. The function described here allows the explicit conversion of the type UDINT into a positive floating point number of the type LREAL in TwinCAT 2 (even if the highest significant bit was set and without compiler warning). Unsigned numbers of the type UDINT are always (implicitly and explicitly) converted into positive floating point numbers in TwinCAT3.

FUNCTION UDINT_TO_LREALEX : LREAL

```
VAR_INPUT
    in      : UDINT;
END_VAR
```

in: Number to be converted.

Sample:

```
PROGRAM MAIN
VAR
    nUdint : UDINT := 16#FFFFFFFF;
    fLreal : LREAL := 0.0;
END_VAR
```

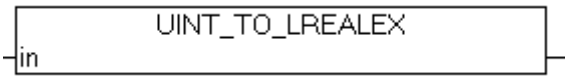
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nUdint	-1, Warning 1105*	+4294967295	+4294967295
fLreal := UDINT_TO_LREAL(nUdint)	-1, Warning 1105*	+4294967295	+4294967295
fLreal := UDINT#16#FFFFFFFF	-1, Warning 1105*	+4294967295	+4294967295
fLreal := 16#FFFFFFFF	+4294967295	+4294967295	+4294967295
fLreal := UDINT_TO _LREALEX(nDword)	+4294967295	+4294967295	+4294967295
fLreal := UDINT_TO _LREALEX(DWORD#16#FFFFFFFF)	+4294967295	+4294967295	+4294967295
fLreal := UDINT_TO _LREALEX(16#FFFFFFFF)	+4294967295	+4294967295	+4294967295

* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.73 UINT_TO_LREALEX



The conversion of unsigned numbers into floating point numbers of the type LREAL is not supported in TwinCAT 2 on the ARM platform. Unsigned numbers with the highest significant bit set may possibly be implicitly converted into negative floating point numbers. The function described here allows the explicit conversion of the type UINT into a positive floating point number of the type LREAL in TwinCAT 2 (even if the highest significant bit was set and without compiler warning). Unsigned numbers of the type UINT are always (implicitly and explicitly) converted into positive floating point numbers in TwinCAT3.

FUNCTION UINT_TO_LREALEX : LREAL

```
VAR_INPUT
    in      : UINT;
END_VAR
```

in: Number to be converted.

Sample:

```
PROGRAM MAIN
VAR
    nUint  : UINT := 16#FFFF;
    fLreal : LREAL := 0.0;
END_VAR
```

fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nUint	+65535, Warning 1105*	+65535	+65535
fLreal := UINT_TO_LREAL(nUint)	+65535, Warning 1105*	+65535	+65535

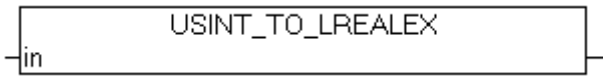
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := UINT#16#FFFF	+65535, Warning 1105*	+65535	+65535
fLreal := 16#FFFF	+65535	+65535	+65535
fLreal := UINT_TO_LREALEX(nUint)	+65535	+65535	+65535
fLreal := UINT_TO_LREALEX(UINT#16#FFFF)	+65535	+65535	+65535
fLreal := UINT_TO_LREALEX(16#FFFF)	+65535	+65535	+65535

* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.74 USINT_TO_LREALEX



The conversion of unsigned numbers into floating point numbers of the type LREAL is not supported in TwinCAT 2 on the ARM platform. Unsigned numbers with the highest significant bit set may possibly be implicitly converted into negative floating point numbers. The function described here allows the explicit conversion of the type USINT into a positive floating point number of the type LREAL in TwinCAT 2 (even if the highest significant bit was set and without compiler warning). Unsigned numbers of the type USINT are always (implicitly and explicitly) converted into positive floating point numbers in TwinCAT3.

FUNCTION USINT_TO_LREALEX : LREAL

```
VAR_INPUT
    in      : USINT;
END_VAR
```

in: Number to be converted.

Sample:

```
PROGRAM MAIN
VAR
    nUsint : USINT := 16#FF;
    fLreal : LREAL := 0.0;
END_VAR
```

fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nUsint	+255, Warning 1105*	+255	+255
fLreal := USINT_TO_LREAL(nUsint)	+255, Warning 1105*	+255	+255
fLreal := USINT#16#FF	+255, Warning 1105*	+255	+255
fLreal := 16#FF	+255	+255	+255
fLreal := USINT_TO_LREALEX(nUsint)	+255	+255	+255
fLreal := USINT_TO_LREALEX(USINT#16#FF)	+255	+255	+255

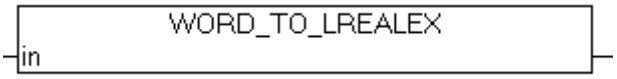
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := USINT_TO_LREALEX(16#FF)	+255	+255	+255

* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.75 WORD_TO_LREALEX



The conversion of unsigned numbers into floating point numbers of the type LREAL is not supported in TwinCAT 2 on the ARM platform. Unsigned numbers with the highest significant bit set may possibly be implicitly converted into negative floating point numbers. The function described here allows the explicit conversion of the type WORD into a positive floating point number of the type LREAL in TwinCAT 2 (even if the highest significant bit was set and without compiler warning). Unsigned numbers of the type WORD are always (implicitly and explicitly) converted into positive floating point numbers in TwinCAT3.

FUNCTION WORD_TO_LREALEX : LREAL

```
VAR_INPUT
    in      : WORD;
END_VAR
```

in: Number to be converted.

Sample:

```
PROGRAM MAIN
VAR
    nWord  : WORD := 16#FFFF;
    fLreal : LREAL := 0.0;
END_VAR
```

fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nWord	+65535, Warning 1105*	+65535	+65535
fLreal := WORD_TO_LREAL(nUsint)	+65535, Warning 1105*	+65535	+65535
fLreal := WORD16#FFFF	+65535, Warning 1105*	+65535	+65535
fLreal := 16#FFFF	+65535	+65535	+65535
fLreal := WORD_TO_LREALEX(nWord)	+65535	+65535	+65535
fLreal := WORD_TO_LREALEX(WORD#16#FFFF)	+65535	+65535	+65535
fLreal := WORD_TO_LREALEX(16#FFFF)	+65535	+65535	+65535

* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

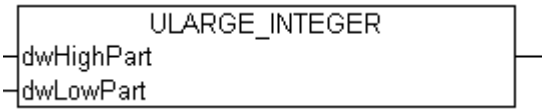
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

H

3.76 Unsigned 64 bit integer

3.76.1 ULARGE_INTEGER



This function initializes/sets the unsigned 64 bit integer value.

FUNCTION ULARGE_INTEGER: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

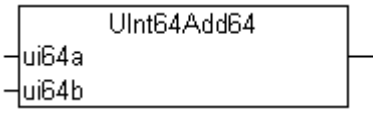
VAR_INPUT

```
VAR_INPUT
    dwHighPart      : DWORD;
    dwLowPart       : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.2 UInt64Add64



The "UInt64Add64" function adds two unsigned 64 Bit integers and returns an unsigned 64-bit integer result.

FUNCTION UInt64Add64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

VAR_INPUT

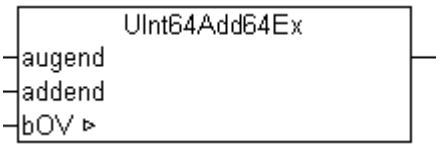
```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030	PC or CX (x86)	TcUtilities.Lib

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1231		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.3 UInt64Add64Ex



Adds two unsigned 64 bit integers and returns an unsigned 64 bit integer result.

FUNCTION UInt64Add64Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

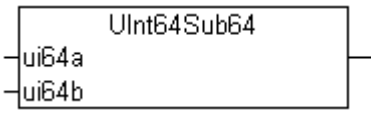
VAR_INPUT

```
VAR_INPUT
    augend : T_ULARGE_INTEGER;
    addend : T_ULARGE_INTEGER;
END_VAR
VAR_IN_OUT
    bOV : BOOL; (* TRUE => arithmetic overflow, FALSE => no overflow *)
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.4 UInt64Sub64



This function will subtract a unsigned 64 bit integer from a unsigned 64 bit integer.

FUNCTION UInt64Sub64 : T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

VAR_INPUT

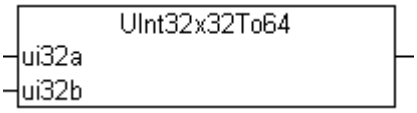
```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Development environment	Target system type	PLC libraries to include
		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.76.5 UInt32x32To64



The function "UInt32x32To64" multiplies two unsigned 32-bit integers, returning an unsigned 64-bit integer result.

FUNCTION UInt32x32To64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

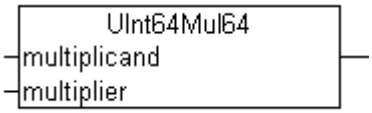
VAR_INPUT

```
VAR_INPUT
    ui32a : DWORD;
    ui32b : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.6 UInt64Mul64



Multiplication of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Mul64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

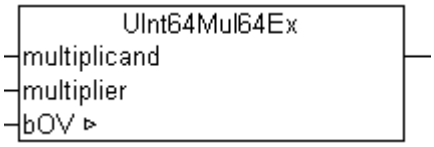
VAR_INPUT

```
VAR_INPUT
    multiplicand: T_ULARGE_INTEGER;
    multiplier: T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.7 UInt64Mul64Ex



Multiplication of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Mul64Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

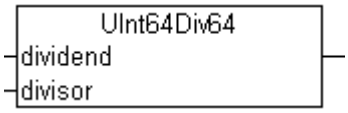
VAR_INPUT

```
VAR_INPUT
    multiplicand : T_ULARGE_INTEGER;
    multiplier   : T_ULARGE_INTEGER;
END_VAR
VAR_IN_OUT
    bOV : BOOL; (* TRUE => Arithmetic overflow, FALSE => no overflow *)
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.8 UInt64Div64



Division of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Div64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

VAR_INPUT

```
VAR_INPUT
    dividend    : T_ULARGE_INTEGER;
    divisor     : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.9 UInt64Div64Ex

Division of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Div64Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

VAR_INPUT

```
VAR_INPUT
    dividend : T_ULARGE_INTEGER;
    divisor  : T_ULARGE_INTEGER;
END_VAR
VAR_IN_OUT
    remainder : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.10 UInt64Div16Ex

The function divides a TwinCAT 2 unsigned 64-bit number (“legacy” type: T_ULARGE_INTEGER [▶ 239]) by a 16-bit unsigned number. The result is an unsigned 64-bit number.

FUNCTION UInt64Div16Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

VAR_INPUT

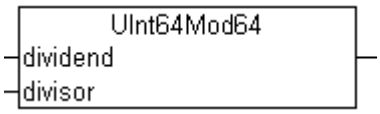
```
VAR_INPUT
    dividend : T_ULARGE_INTEGER;
    divisor  : WORD;
END_VAR
VAR_IN_OUT
    remainder : T_ULARGE_INTEGER;
END_VAR
```

Requirements

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

Development environment	Target system type	PLC libraries to include
		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.76.11 UInt64Mod64



Modulo-Division of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Mod64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

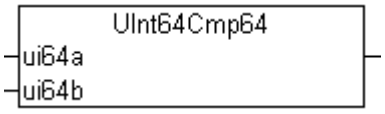
VAR_INPUT

```
VAR_INPUT
    dividend : T_ULARGE_INTEGER;
    divisor  : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.12 UInt64Cmp64



This function compares two unsigned 64 bit integers.

FUNCTION UInt64Cmp64: DINT

VAR_INPUT

```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Return value	Description
-1	ui64a is less than *ui64b*
0	ui64a is equal to *ui64b*
1	ui64a is greater than *ui64b*

Requirements

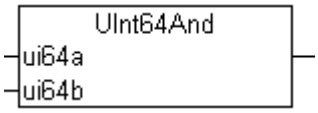
Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030	PC or CX (x86)	TcUtilities.Lib

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1231		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Also see about this

T_ULARGE_INTEGER [▶ 239]

3.76.13 UInt64And



Bitwise AND of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64And: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

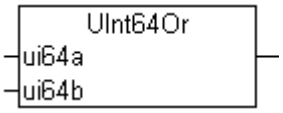
VAR_INPUT

```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.14 UInt64Or



Bitwise OR of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Or: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

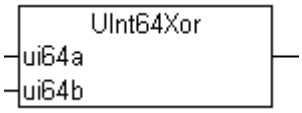
VAR_INPUT

```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.15 UInt64Xor



Bitwise XOR of two unsigned 64 bit integers. The result is an unsigned 64 bit integer.

FUNCTION UInt64Xor: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

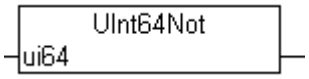
VAR_INPUT

```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.16 UInt64Not



Bitwise NOT of unsigned 64 bit integer. The result is an unsigned 64 bit integer.

FUNCTION UInt64Not: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

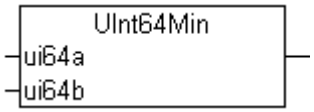
VAR_INPUT

```
VAR_INPUT
    ui64 : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.17 UInt64Min



Minimum function. Returns the lesser of the two values.

FUNCTION UInt64Min: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

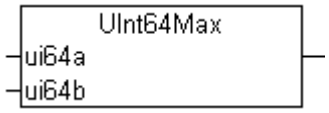
VAR_INPUT

```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.18 UInt64Max



Maximum function. Returns the greater of the two values.

FUNCTION UInt64Max: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

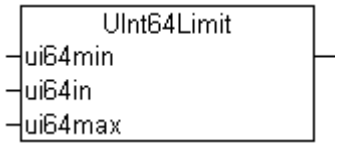
VAR_INPUT

```
VAR_INPUT
    ui64a : T_ULARGE_INTEGER;
    ui64b : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.19 UInt64Limit



Limiting function. The result is unsigned 64 bit integer.

FUNCTION UInt64Limit: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

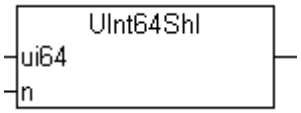
VAR_INPUT

```
VAR_INPUT
    ui64min : T_ULARGE_INTEGER;
    ui64in  : T_ULARGE_INTEGER;
    ui64max : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.20 UInt64Shl



Bitwise left-shift of an operand.

FUNCTION UInt64Shl: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

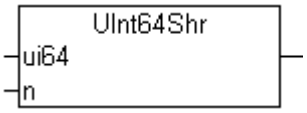
VAR_INPUT

```
VAR_INPUT
    ui64    : T_ULARGE_INTEGER;
    n       : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.21 UInt64Shr



Bitwise right-shift of an operand.

FUNCTION UInt64Shr: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

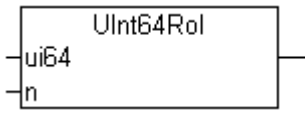
VAR_INPUT

```
VAR_INPUT
    ui64    : T_ULARGE_INTEGER;
    n       : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.22 UInt64Rol



Bitwise rotation of an operand to the left.

FUNCTION UInt64Rol: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

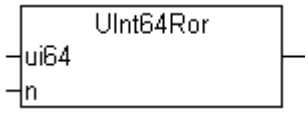
VAR_INPUT

```
VAR_INPUT
    ui64    : T_ULARGE_INTEGER;
    n       : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.23 UInt64Ror



Bitwise rotation of an operand to the right.

FUNCTION UInt64Ror: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

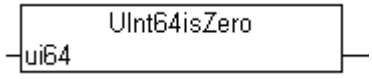
VAR_INPUT

```
VAR_INPUT
    ui64    : T_ULARGE_INTEGER;
    n       : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.24 UInt64isZero



This function returns TRUE if value of the unsigned 64 bit integer variable is zero.

FUNCTION UInt64isZero: BOOL

VAR_INPUT

```
VAR_INPUT
    ui64 : T_ULARGE_INTEGER;
END_VAR
```

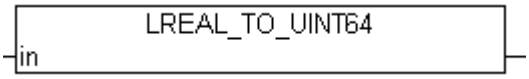
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Also see about this

T_ULARGE_INTEGER [▶ 239]

3.76.25 LREAL_TO_UINT64



Converts LREAL number to unsigned 64 bit number.

FUNCTION LREAL_TO_UINT64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

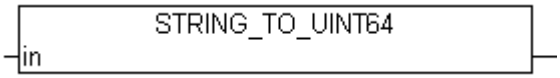
VAR_INPUT

```
VAR_INPUT
    in      : LREAL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.26 STRING_TO_UINT64



Converts STRING to unsigned 64 bit number

FUNCTION STRING_TO_UINT64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

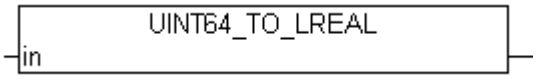
VAR_INPUT

```
VAR_INPUT
    in : STRING(21);
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.27 UINT64_TO_LREAL



Converts unsigned 64 bit number to LREAL.

FUNCTION UINT64_TO_LREAL: LREAL

VAR_INPUT

```
VAR_INPUT
    in : T_ULARGE_INTEGER;
END_VAR
```

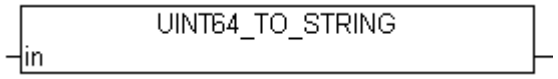
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Also see about this

T_ULARGE_INTEGER [▶ 239]

3.76.28 UINT64_TO_STRING



Converts unsigned 64 bit number to STRING.

FUNCTION UINT64_TO_STRING: STRING(21)

VAR_INPUT

```
VAR_INPUT
    in : T_ULARGE_INTEGER;
END_VAR
```

Requirements

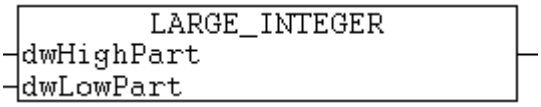
Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Also see about this

T_ULARGE_INTEGER [▶ 239]

3.77 Signed 64 bit integer

3.77.1 LARGE_INTEGER



This function initializes/sets the signed 64 bit integer value.

FUNCTION LARGE_INTEGER: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

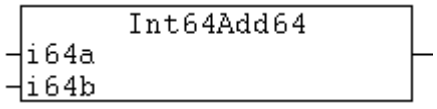
VAR_INPUT

```
VAR_INPUT
    dwHighPart      : DWORD;
    dwLowPart       : DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.2 Int64Add64



The "Int64Add64" function adds two signed 64 Bit integers and returns an signed 64-bit integer result.

FUNCTION Int64Add64: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

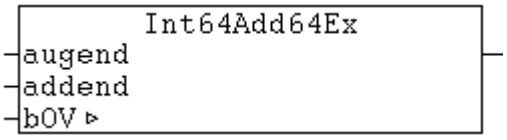
VAR_INPUT

```
VAR_INPUT
    i64a : T_LARGE_INTEGER;
    i64b : T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.3 Int64Add64Ex



Adds two signed 64 bit integers and returns an signed 64 bit integer result.

FUNCTION Int64Add64Ex: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

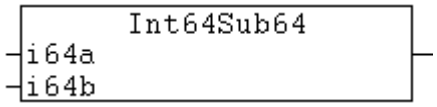
VAR_INPUT

```
VAR_INPUT
    augend : T_LARGE_INTEGER;
    addend : T_LARGE_INTEGER;
END_VAR
VAR_IN_OUT
    bOV : BOOL; (* TRUE => arithmetic overflow, FALSE => no overflow *)
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.4 Int64Sub64



This function will subtract a signed 64 bit integer from a signed 64 bit integer.

FUNCTION Int64Sub64 : T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

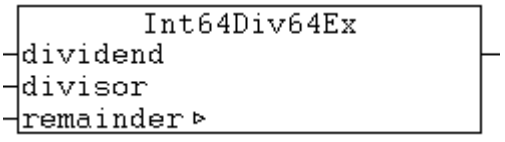
VAR_INPUT

```
VAR_INPUT
    i64a : T_LARGE_INTEGER;
    i64b : T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.5 Int64Div64Ex



Division of two signed 64 bit integers. The result is an signed 64 bit integer.

FUNCTION Int64Div64Ex: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

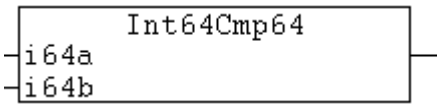
VAR_INPUT

```
VAR_INPUT
    dividend : T_LARGE_INTEGER;
    divisor : T_LARGE_INTEGER;
END_VAR
VAR_IN_OUT
    remainder : T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.6 Int64Cmp64



This function compares two signed 64 bit integers.

FUNCTION Int64Cmp64: DINT

VAR_INPUT

```
VAR_INPUT
    i64a : T_LARGE_INTEGER;
    i64b : T_LARGE_INTEGER;
END_VAR
```

Return value	Description
-1	i64a is less than *i64b*
0	i64a is equal to *i64b*
1	i64a is greater than *i64b*

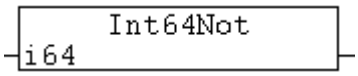
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

T_LARGE_INTEGER [▶ 240]

3.77.7 Int64Not



Bitwise NOT of signed 64 bit integer. The result is an signed 64 bit integer.

FUNCTION Int64Not: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

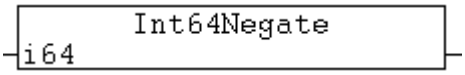
VAR_INPUT

```
VAR_INPUT
    i64 : T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.8 Int64Negate



This function negates signed 64 bit integer (returns 1's complement of signed 64 bit integer).

FUNCTION Int64Negate: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

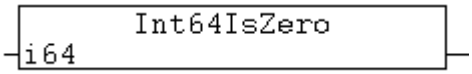
VAR_INPUT

```
VAR_INPUT
    i64 : T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.9 Int64isZero



This function returns TRUE if value of the signed 64 bit integer variable is zero.

FUNCTION Int64isZero: BOOL

VAR_INPUT

```
VAR_INPUT
    i64 : T_LARGE_INTEGER;
END_VAR
```

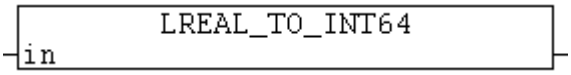
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_LARGE_INTEGER [▶ 240]

3.77.10 LREAL_TO_INT64



Converts LREAL number to signed 64 bit number.

FUNCTION LREAL_TO_INT64: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

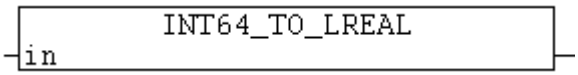
VAR_INPUT

```
VAR_INPUT
    in      : LREAL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.11 INT64_TO_LREAL



Converts signed 64 bit number to LREAL.

FUNCTION INT64_TO_LREAL: LREAL

VAR_INPUT

```
VAR_INPUT
    in : T_LARGE_INTEGER;
END_VAR
```

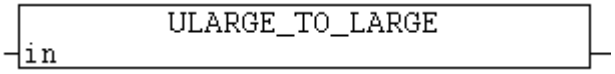
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_LARGE_INTEGER [▶ 240]

3.77.12 ULARGE_TO_LARGE



Converts unsigned 64 bit number to signed 64 bit number.

FUNCTION ULARGE_TO_LARGE: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

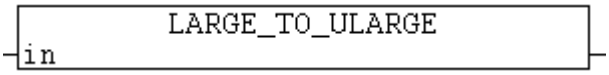
VAR_INPUT

```
VAR_INPUT
    in : T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.13 LARGE_TO_ULARGE



Converts signed 64 bit number to unsigned 64 bit number.

FUNCTION LARGE_TO_ULARGE: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

VAR_INPUT

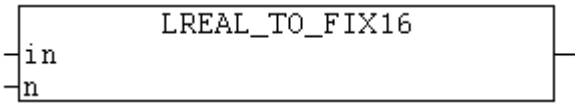
```
VAR_INPUT
    in : T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.78 Signed 16 bit fixed point

3.78.1 LREAL_TO_FIX16



Converts a floating-point number of type: LREAL to a signed 16 bit fixed-point number with the required number of decimal places.

FUNCTION LREAL_TO_FIX16 : T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    in : LREAL;
    n  : WORD(0..15) := 15;
END_VAR
```

in: The LREAL number to be converted.

n: Number of required decimal places.

Example:

In the following example several constants are converted to fixed-point numbers. The number of decimal places can be specified for the conversion. Please note that (similar to the conversion of floating-point numbers) rounding errors may occur (q2 and q15 in our example).

```
PROGRAM TEST
VAR
    q2, q4, q8, q12, q15 : T_FIX16;
    r2, r4, r8, r12, r15 : LREAL;
END_VAR
```

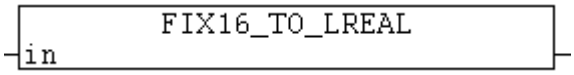
```
q2 := LREAL_TO_FIX16( 0.6, 2 );
q4 := LREAL_TO_FIX16( -0.25, 4 );
q8 := LREAL_TO_FIX16( -0.75, 8 );
q12 := LREAL_TO_FIX16( 2.30078125, 12 );
q15 := LREAL_TO_FIX16( 0.6, 15 );
```

```
r2 := FIX16_TO_LREAL( q2 );      (* 0.5 *)
r4 := FIX16_TO_LREAL( q4 );       (* -0.25 *)
r8 := FIX16_TO_LREAL( q8 );      (* -0.75 *)
r12 := FIX16_TO_LREAL( q12 );    (* 2.30078125 *)
r15 := FIX16_TO_LREAL( q15 );    (* 0.600006103515625 *)
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.2 FIX16_TO_LREAL



Converts a signed 16-bit fixed-point number to a floating-point number of type: LREAL.

FUNCTION FIX16_TO_LREAL : LREAL

VAR_INPUT

```
VAR_INPUT
    in : T_FIX16;
END_VAR
```

in: The fixed-point number [▶ 240] to be converted.

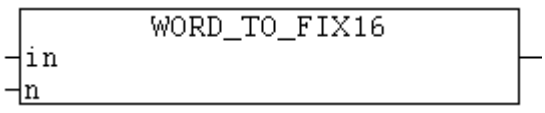
Example:

See function description: LREAL_TO_FIX16 [▶ 98].

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.3 WORD_TO_FIX16



The function converts a WORD variable to a 16-bit fixed-point number (the WORD variable contains the coded digits and decimal places for the fixed-point number).

FUNCTION WORD_TO_FIX16 : T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    in : WORD; (* 16 bit fixed point number *)
    n  : WORD(0..15);(* number of fractional bits *)
END_VAR
```

Example:

```
PROGRAM WORD_TO_FIX
VAR
    double: LREAL;
END_VAR
```

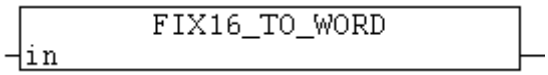
```
double := FIX16_TO_LREAL(WORD_TO_FIX16(2#0000110010000000, 8));
```

The value of the *double* variable is: 12.5

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.4 FIX16_TO_WORD



This function converts a 16-bit fixed-point number to a WORD variable (the WORD variable contains the digits and decimal places for the fixed-point number).

FUNCTION FIX16_TO_WORD : WORD

VAR_INPUT

```
VAR_INPUT
    in      : T_FIX16; (* 16 bit fixed point number *)
END_VAR
```

Example:

```
PROGRAM FIX_TO_WORD
VAR
    fp16 : WORD;
END_VAR
```

```
fp16 := FIX16_TO_WORD(LREAL_TO_FIX16(12.5, 8));
```

The value of the *fp16* variable is: 2#0000110010000000.

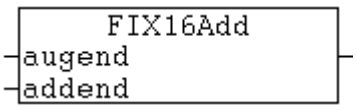
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_FIX16 [▶ 240]

3.78.5 FIX16Add



This function adds two signed 16-bit fixed-point numbers. The numbers do not have to have the same resolution (number of decimal places). The resolution of the number with the higher number of decimal places is reduced before the addition, i.e. the decimal places of the number with the higher resolution are truncated. The result of the addition is a signed 16-bit fixed-point number.

FUNCTION FIX16Add : T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    augend : T_FIX16;
    addend : T_FIX16;
END_VAR
```

augend: The first summand [▶ 240].

addend: The second summand.

Example:

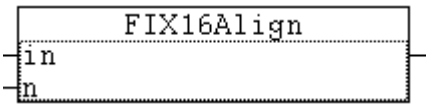
```
PROGRAM FIXADD
VAR
    a, b : T_FIX16;
    result : LREAL;
END_VAR
```

```
a := LREAL_TO_FIX16( 0.5, 8 );
b := LREAL_TO_FIX16( -0.25, 8 );

result := FIX16_TO_LREAL( FIX16Add( a, b ) ); (* The result is: 0.25 *)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.6 FIX16Align



This function can be used to change the resolution (number of decimal places) of a signed 16-bit fixed-point number. The function supplies the new fixed-point number as return parameter.

FUNCTION FIX16Align: T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    in : T_FIX16;
    n  : BYTE(0..15);
END_VAR
```

in: Fixed-point number [▶ 240] whose resolution is to be modified.

n: The new number of decimal places.

Example:

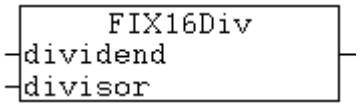
```
PROGRAM FIXALIGN
VAR
    q8, q4 : T_FIX16;
    result : LREAL;
END_VAR
```

```
q8 := LREAL_TO_FIX16( 0.6, 8 );
result := FIX16_TO_LREAL( q8 );(* The result is: 0.6015625 *)

q4 := FIX16Align( q8, 4 );
result := FIX16_TO_LREAL( q4 );(* The result is: 0.5625 *)
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.7 FIX16Div



This function divides two signed 16-bit fixed-point numbers. The numbers do not have to have the same resolution (number of decimal places). The resolution of the number with the higher number of decimal places is reduced before the division. i.e. the decimal places of the number with the higher resolution are truncated. The result of the division is a signed 16-bit fixed-point number.

FUNCTION FIX16Div: T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    dividend : T_FIX16;
    divisor : T_FIX16;
END_VAR
```

dividend: Number that is divided [▶ 240].

divisor: Number by which the dividend is divided.

Example:

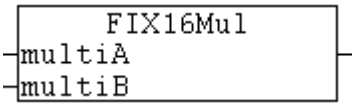
```
PROGRAM FIXDIV
VAR
    a, b : T_FIX16;
    result : LREAL;
END_VAR
```

```
a := LREAL_TO_FIX16( -22.5, 8 );
b := LREAL_TO_FIX16( 10.0, 8 );

result := FIX16_TO_LREAL( FIX16Div( a, b ) ); (* The result is: -2.25 *)
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.8 FIX16Mul



This function multiplies two signed 16-bit fixed-point numbers. The numbers do not have to have the same resolution (number of decimal places). The resolution of the number with the higher number of decimal places is reduced before the multiplication, i.e. the decimal places of the number with the higher resolution are truncated. The result of the multiplication is a signed 16-bit fixed-point number.

FUNCTION FIX16Mul: T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    multiA : T_FIX16;
    multiB : T_FIX16;
END_VAR
```

multiA: The first multiplier [▶ 240].

multiB: The second multiplier.

Example:

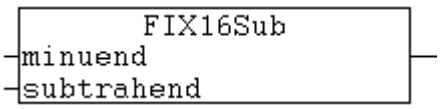
```
PROGRAM FIXMUL
VAR
    a, b : T_FIX16;
    result : LREAL;
END_VAR
```

```
a := LREAL_TO_FIX16( 0.25, 8 );
b := LREAL_TO_FIX16( 10.0, 8 );

result := FIX16_TO_LREAL( FIX16Mul( a, b ) ); (* The result is: 2.5 *)
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.9 FIX16Sub



This function subtracts two signed 16-bit fixed-point numbers. The numbers do not have to have the same resolution (number of decimal places). The resolution of the number with the higher number of decimal places is reduced before the subtraction, i.e. the decimal places of the number with the higher resolution are truncated. The result of the subtraction is a signed 16-bit fixed-point number.

FUNCTION FIX16Sub : T_FIX16

T_FIX16 [▶ 240]

VAR_INPUT

```
VAR_INPUT
    minuend    : T_FIX16;
    subtrahend : T_FIX16;
END_VAR
```

minuend: Number [▶ 240] from which a value is subtracted.

subtrahend: Number that is subtracted.

Example:

```
PROGRAM FIXSUB
VAR
    a, b : T_FIX16;
    result : LREAL;
END_VAR
```

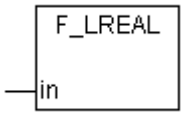
```
a := LREAL_TO_FIX16( 0.5, 8 );
b := LREAL_TO_FIX16( 0.75, 8 );

result := FIX16_TO_LREAL( FIX16Sub( a, b ) ); (* The result is: -0.25 *)
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.79 Helper functions

3.79.1 F_LREAL



Helper function, returns struct variable with additional LREAL variable information.

FUNCTION F_LREAL : T_Arg

T_Arg [▶ 239]

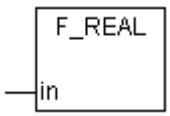
VAR_IN_OUT

```
VAR_IN_OUT
    in      :LREAL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.2 F_REAL



Helper function, returns struct variable with additional REAL variable information.

FUNCTION F_REAL : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :REAL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.3 F_BYTE

This is an auxiliary function that returns information about a BYTE variable with a certain structure.

FUNCTION F_BYTE : T_Arg

T_Arg [▶ 239]

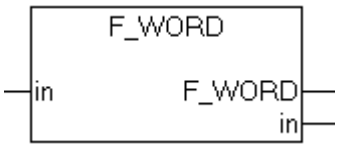
VAR_IN_OUT

```
VAR_IN_OUT
    in      :BYTE;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.4 F_WORD



This is an auxiliary function that returns information about a WORD variable with a certain structure.

FUNCTION F_WORD : T_Arg

T_Arg [▶ 239]

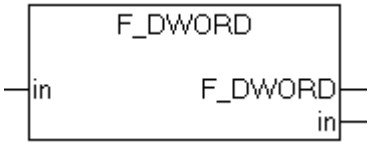
VAR_IN_OUT

```
VAR_IN_OUT
    in      :WORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.5 F_DWORD



This is an auxiliary function that returns information about a DWORD variable with a certain structure.

FUNCTION F_DWORD : T_Arg

T_Arg [▶ 239]

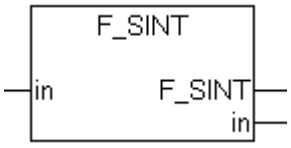
VAR_IN_OUT

```
VAR_IN_OUT
    in      :DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.6 F_SINT



This is an auxiliary function that returns information about a SINT variable with a certain structure.

FUNCTION F_SINT : T_Arg

T_Arg [▶ 239]

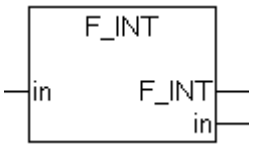
VAR_IN_OUT

```
VAR_IN_OUT
    in      :SINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.7 F_INT



This is an auxiliary function that returns information about an INT variable with a certain structure.

FUNCTION F_INT : T_Arg

T_Arg [▶ 239]

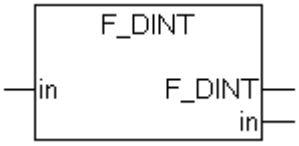
VAR_IN_OUT

```
VAR_IN_OUT
    in      :INT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.8 F_DINT



This is an auxiliary function that returns information about a DINT variable with a certain structure.

FUNCTION F_DINT : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

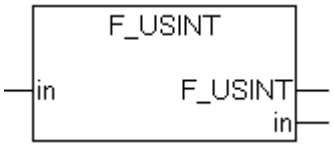
```
VAR_IN_OUT
    in      :DINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Development environment	Target system type	PLC libraries to include
		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.79.9 F_USINT



This is an auxiliary function that returns information about a USINT variable with a certain structure.

FUNCTION F_USINT : T_Arg

T_Arg [▶ 239]

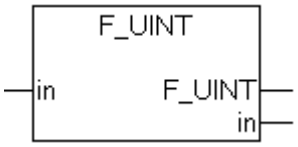
VAR_IN_OUT

```
VAR_IN_OUT
    in      :USINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.10 F_UINT



This is an auxiliary function that returns information about a UDINT variable with a certain structure.

FUNCTION F_UINT : T_Arg

T_Arg [▶ 239]

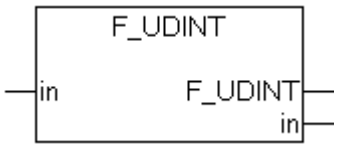
VAR_IN_OUT

```
VAR_IN_OUT
    in      :UINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.11 F_UDINT



This is an auxiliary function that returns information about a UDINT variable with a certain structure.

FUNCTION F_UDINT : T_Arg

T_Arg [▶ 239]

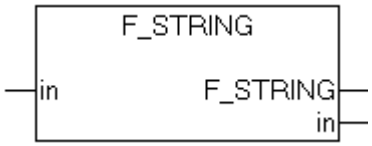
VAR_IN_OUT

```
VAR_IN_OUT
    in      :UDINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.12 F_STRING



This is an auxiliary function that returns information about a T_MaxString variable with a certain structure.

FUNCTION F_STRING : T_Arg

T_Arg [▶ 239]

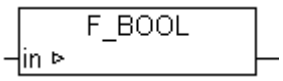
VAR_IN_OUT

```
VAR_IN_OUT
    in      :T_MaxString;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.13 F_BOOL



Helper function, returns struct variable with additional BOOL variable information.

FUNCTION F_BOOL : T_Arg

T_Arg [▶ 239]

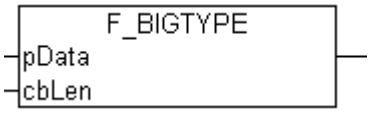
VAR_IN_OUT

```
VAR_IN_OUT
    in      :BOOL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.14 F_BIGTYPE



Helper function, returns struct variable with additional struct or array variable information.

FUNCTION F_BIGTYPE: T_Arg

T_Arg [▶ 239]

VAR_INPUT

```
VAR_INPUT
    pData   : DWORD;
    cbLen   : DWORD;
END_VAR
```

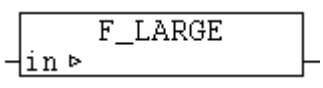
pData: Pointer to data (returned by ADR operator).

cbLen: Data byte length (returned by SIZEOF operator).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1235	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.15 F_LARGE



This is an auxiliary function that returns information about a T_LARGE_INTEGER variable with a certain structure.

FUNCTION F_LARGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :T_LARGE_INTEGER;
END_VAR
```

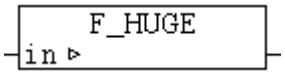
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_LARGE_INTEGER [▶ 240]

3.79.16 F_HUGE



This is an auxiliary function that returns information about a T_HUGE_INTEGER variable with a certain structure.

FUNCTION F_HUGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :T_HUGE_INTEGER;
END_VAR
```

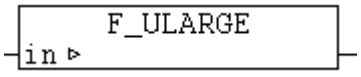
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_HUGE_INTEGER [▶ 240]

3.79.17 F_ULARGE



This is an auxiliary function that returns information about a T_ULARGE_INTEGER variable with a certain structure.

FUNCTION F_ULARGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :T_ULARGE_INTEGER;
END_VAR
```

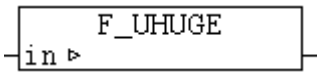
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_ULARGE_INTEGER [▶ 239]

3.79.18 F_UHUGE



This is an auxiliary function that returns information about a T_UHUGE_INTEGER variable with a certain structure.

FUNCTION F_UHUGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :T_UHUGE_INTEGER;
END_VAR
```

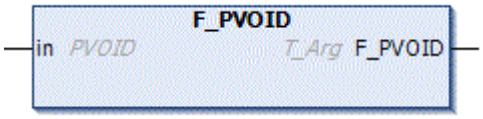
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_UHUGE_INTEGER [▶ 240]

3.79.19 F_PVOID



Help function that returns a structure with information (type: T_Arg [▶ 239]) on a BYTE variable.

FUNCTION F_PVOID : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :PVOID;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

3.79.20 F_ARGCPY

This function copies the value of a variable of type T_Arg into another variable and supplies the number of successfull copied databytes as return value.

FUNCTION F_ARGCPY: UDINT

VAR_INPUT

```
VAR_INPUT
    typeSafe: BOOL;
END_VAR
```

typeSafe: If TRUE => Only equal types can be compared (type safe compare). FALSE => Different types can be compared(type independent compare).

VAR_IN_OUT

```
VAR_IN_OUT
    dest : T_Arg;
    src : T_Arg;
END_VAR
```

dest: Target variable [▶ 239] to be copied into.

src: Source variable to be copied.

Return parameter	Meaning
0	False parameter values, type, length, value of *adest* or *src* == 0
> 0	On success, the number of copied bytes

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.79.21 F_ARGCMP

This function compares two variables of type T_Arg and supplies the result as return parameter.

FUNCTION F_ARGCMP: DINT

VAR_INPUT

```
VAR_INPUT
    typeSafe: BOOL;
    arg1 : T_Arg;
    arg2 : T_Arg;
END_VAR
```

typeSafe: If TRUE => Only equal types can be compared (type safe compare). FALSE => Different types can be compared(type independent compare).

arg1: First variable [▶ 239] to be compared.

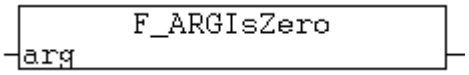
arg2:Second variable to be compared.

Return parameter	Relation of the first different byte (type, length, value) in the first and second variable
-3	Length of *arg1* less than *arg2*
-2	Type of *arg1* less than *arg2*
-1	Value of von *arg*1 less than *arg2*
0	arg1 equal with *arg2*
1	Value of *arg1* greater than *arg2*
2	Type of *arg1* greater than *arg2*
3	Length of *arg1* greater than *arg2*
0xFF	False parameter values, type, length, value of *arg1 or arg2* = 0

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.79.22 F_ARGISZERO



This function returns TRUE if the value one of the T_Arg member is zero or not initialized.

FUNCTION F_ARGISZERO: BOOL

VAR_INPUT

```
VAR_INPUT
    arg : T_Arg;
END_VAR
```

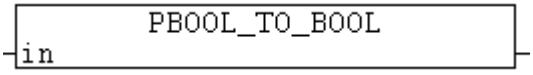
arg: Variable [▶ 239] to be checked.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.80 P[TYPE]_TO_[TYPE] converting functions

3.80.1 PBOOL_TO_BOOL



This function returns the content (value) of BOOL pointer identifier.

FUNCTION PBOOL_TO_BOOL: BOOL

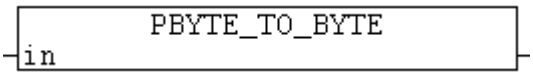
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO BOOL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.2 PBYTE_TO_BYTE



This function returns the content (value) of BYTE pointer identifier.

FUNCTION PBYTE_TO_BYTE: BYTE

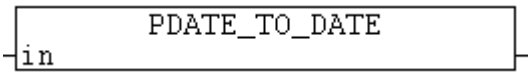
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO BYTE;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.3 PDATE_TO_DATE



This function returns the content (value) of DATE pointer identifier.

FUNCTION PDATE_TO_DATE: DATE

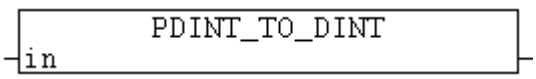
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO DATE;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.4 PDINT_TO_DINT



This function returns the content (value) of DINT pointer identifier.

FUNCTION PDINT_TO_DINT: DINT

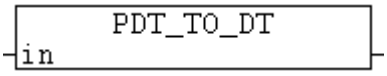
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO DINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.5 PDT_TO_DT



This function returns the content (value) of DT pointer identifier.

FUNCTION PDT_TO_DT: DT

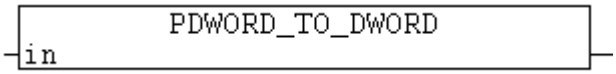
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO DT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.6 PDWORD_TO_DWORD



This function returns the content (value) of DWORD pointer identifier.

FUNCTION PDWORD_TO_DWORD: DWORD

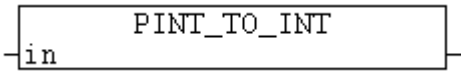
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO DWORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.7 PINT_TO_INT



This function returns the content (value) of INT pointer identifier.

FUNCTION PINT_TO_INT: INT

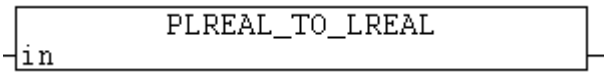
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO INT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.8 PLREAL_TO_LREAL



This function returns the content (value) of PLREAL pointer identifier.

FUNCTION PLREAL_TO_LREAL: LREAL

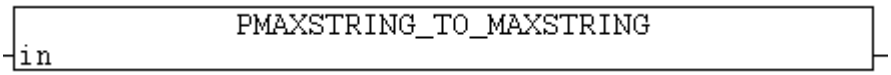
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO LREAL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.9 PMAXSTRING_TO_MAXSTRING



This function returns the content (value) of T_MaxString pointer identifier.

FUNCTION PMAXSTRING_TO_MAXSTRING: T_MaxString

T_MaxString

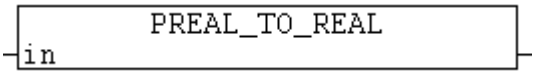
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO T_MaxString;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.10 PREAL_TO_REAL



This function returns the content (value) of REAL pointer identifier.

FUNCTION PREAL_TO_REAL: REAL

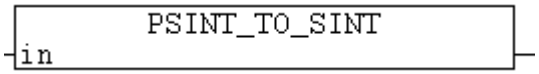
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO REAL;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.11 PSINT_TO_SINT



This function returns the content (value) of SINT pointer identifier.

FUNCTION PSINT_TO_SINT: SINT

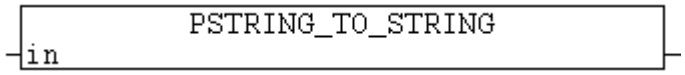
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO SINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.12 PSTRING_TO_STRING



This function returns the content (value) of STRING pointer identifier.

FUNCTION PSTRING_TO_STRING: STRING

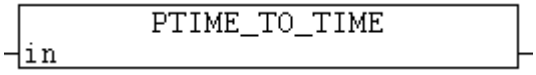
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO STRING;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.13 PTIME_TO_TIME



This function returns the content (value) of TIME pointer identifier.

FUNCTION PTIME_TO_TIME: TIME

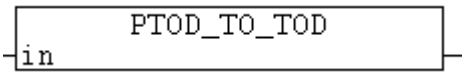
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO TIME;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.14 PTOD_TO_TOD



This function returns the content (value) of TOD pointer identifier.

FUNCTION PTOD_TO_TOD: TOD

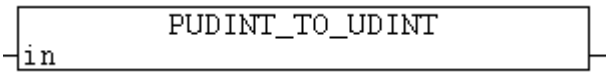
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO TOD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.15 PUDINT_TO_UDINT



This function returns the content (value) of UDINT pointer identifier.

FUNCTION PUDINT_TO_UDINT: UDINT

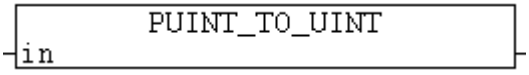
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO UDINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.16 PUINT_TO_UINT



This function returns the content (value) of UINT pointer identifier.

FUNCTION PUINT_TO_UINT: UINT

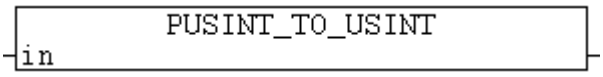
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO UINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.17 PUSINT_TO_USINT



This function returns the content (value) of USINT pointer identifier.

FUNCTION PUSINT_TO_USINT: USINT

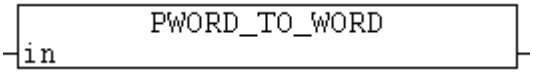
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO USINT;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.18 PWORD_TO_WORD



This function returns the content (value) of WORD pointer identifier.

FUNCTION PWORD_TO_WORD: WORD

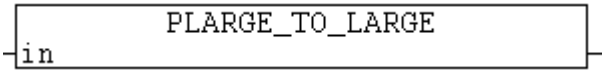
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO WORD;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.19 PLARGE_TO_LARGE



This function returns the content (value) of T_LARGE_INTEGER pointer identifier.

FUNCTION PLARGE_TO_LARGE: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

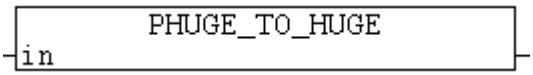
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO T_LARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.20 PHUGE_TO_HUGE



This function returns the content (value) of T_HUGE_INTEGER pointer identifier.

FUNCTION PHUGE_TO_HUGE: T_HUGE_INTEGER

T_HUGE_INTEGER [▶ 240]

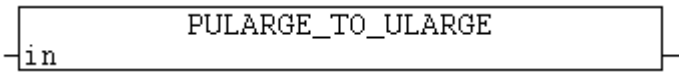
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO T_HUGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.21 PULARGE_TO_ULARGE



This function returns the content (value) of T_ULARGE_INTEGER pointer identifier.

FUNCTION PULARGE_TO_ULARGE: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

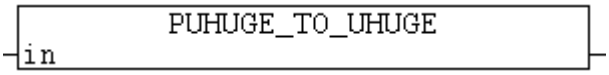
VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO T_ULARGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.22 PUHUGE_TO_UHUGE



This function returns the content (value) of T_UHUGE_INTEGER pointer identifier.

FUNCTION PUHUGE_TO_UHUGE: T_UHUGE_INTEGER

T_UHUGE_INTEGER [▶ 240]

VAR_INPUT

```
VAR_INPUT
    in      : POINTER TO T_UHUGE_INTEGER;
END_VAR
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.81 Byte order converting functions

3.81.1 Host Byte Order / Network Byte Order

The byte order is fixed in network protocols. This is called the Network Byte Order. The natural byte order in the TwinCAT system is called the Host Byte Order. In most cases, the required Network Byte Order corresponds to the Big Endian Format (MOTOROLA). However, the TwinCAT PLC system uses the Little Endian Format (Intel). So that the error-free exchange of data between the TwinCAT PLC system and a different platform can take place, the byte order in the application program must be converted accordingly.

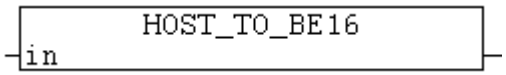
Data that is to be transmitted via a network protocol from the TwinCAT system (host) to an external system can be converted to the network format using the following functions:

- HOST_TO_BE16 [▶ 122]
- HOST_TO_BE32 [▶ 123]
- HOST_TO_BE64 [▶ 123]
- HOST_TO_BE128 [▶ 124]

Conversely, the received network data (external system) can be converted to the host format (TwinCAT system) using the following functions:

- BE16_TO_HOST [▶ 124]
- BE32_TO_HOST [▶ 124]
- BE64_TO_HOST [▶ 125]
- BE128_TO_HOST [▶ 125]

3.81.2 HOST_TO_BE16



This function converts host (little endian, intel) 16 bit number to network (big endian, motorola) format. See also: Byte Order [▶ 122].

FUNCTION HOST_TO_BE16 : WORD

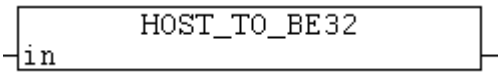
```
VAR_INPUT
    in      : WORD;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.3 HOST_TO_BE32



This function converts host (little endian, intel) 32 bit number to network (big endian, motorola) format. See also: Byte Order [▶ 122].

FUNCTION HOST_TO_BE32 : DWORD

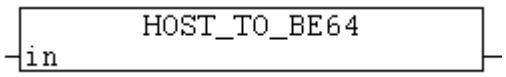
```
VAR_INPUT
    in      : DWORD;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.4 HOST_TO_BE64



This function converts host (little endian, intel) 64 bit number to network (big endian, motorola) format. See also: Byte Order [▶ 122].

FUNCTION HOST_TO_BE64 : T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

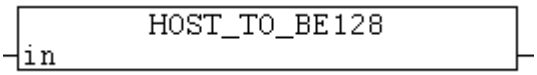
```
VAR_INPUT
    in      : T_ULARGE_INTEGER;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.5 HOST_TO_BE128



This function converts host (little endian, intel) 128 bit number to network (big endian, motorola) format. See also: Byte Order [▶ 122].

FUNCTION HOST_TO_BE128 : T_UHUGE_INTEGER

T_UHUGE_INTEGER [▶ 240]

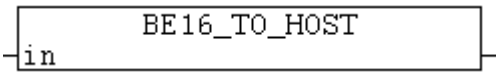
```
VAR_INPUT
    in      : T_UHUGE_INTEGER;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.6 BE16_TO_HOST



This function converts network (big endian, motorola) 16 bit number to host (little endian, intel) format. See also: Byte Order [▶ 122].

FUNCTION BE16_TO_HOST : WORD

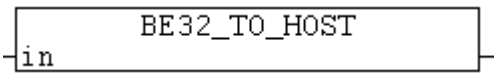
```
VAR_INPUT
    in      : WORD;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.7 BE32_TO_HOST



This function converts network (big endian, motorola) 32 bit number to host (little endian, intel) format. See also: Byte Order [▶ 122].

FUNCTION BE32_TO_HOST : DWORD

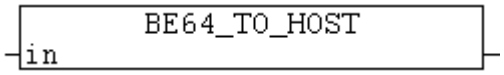
```
VAR_INPUT
    in      : DWORD;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.8 BE64_TO_HOST



This function converts network (big endian, motorola) 64 bit number to host (little endian, intel) format. See also: Byte Order [▶ 122].

FUNCTION BE64_TO_HOST : T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 239]

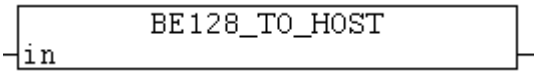
```
VAR_INPUT
    in      : T_ULARGE_INTEGER;
END_VAR
```

in: Number to be converted.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.9 BE128_TO_HOST



This function converts network (big endian, motorola) 128 bit number to host (little endian, intel) format. See also: Byte Order [▶ 122].

FUNCTION BE128_TO_HOST : T_UHUGE_INTEGER

T_UHUGE_INTEGER [▶ 240]

```
VAR_INPUT
    in      : T_UHUGE_INTEGER;
END_VAR
```

in: Number to be converted.

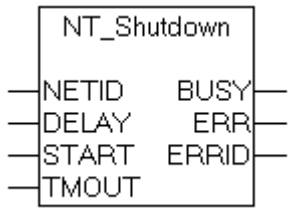
Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

4 Function blocks

4.1 NT_Shutdown

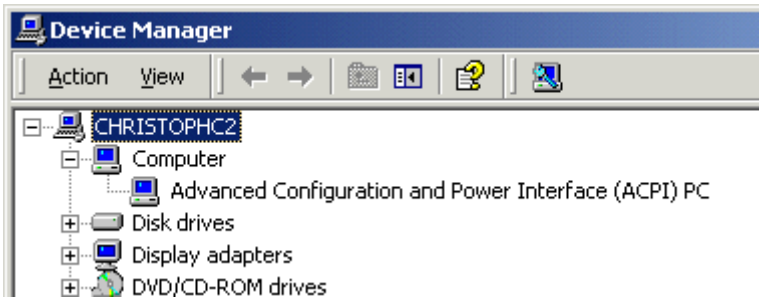
This functionality is not available under Windows CE!



The Windows NT operating system can be shut down with the aid of the "NT_Shutdown" function block. The function largely corresponds to the Shut Down command on the Windows taskbar. A delay before execution of the Shut Down command can be defined via the DELAY parameter. Internally, an instance of the ADSWRTCTRL function block is called.

Requirements:

Operating systems (up to Windows 2000 and higher) perform with the aid of the "NT_Shutdown" function block the "Shutdown with Power OFF" (the computer switches its power OFF). This function can only be used on systems which are ACPI conform (Advanced Configuration and Power Interface). The ACPI functions should be activated in BIOS before the installation of the operating system.



The default TwinCAT settings are to perform shutdown with power OFF. You can disable the power OFF function in windows registry. Please add following entry (implemented in TwinCAT Version >= 2.7 Build >= 505):

"DisableACPIPowerOff" REG_DWORD = 0x00000001 in Registry under:
"HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System"

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    DELAY       :DWORD;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the operating system is to be shut down. If it is the local computer that is to be shut down, an empty string can be entered.

DELAY: The delay time, in seconds, before the Shut Down command is executed.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

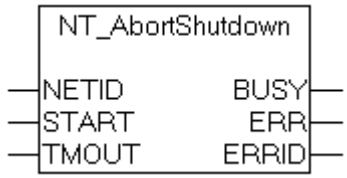
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

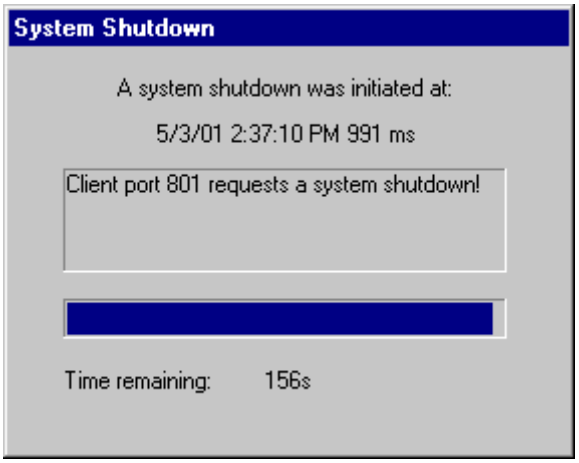
4.2 NT_AbortShutdown



The "NT_AbortShutdown" function block can be used to abort a shutdown command that has previously been called with the NT_Shutdown [▶ 126] function block. Internally, an instance of the ADSWRTCTRL function block is called.

Comment:

When calling the NT_Shutdown [▶ 126] function block, a delay period can be given as a parameter. The remaining time is indicated in a message window.



Only after the delay time has elapsed is the operating system shut down. During this time, the shutdown process can be interrupted from the PLC with the aid of the "NT_AbortShutdown" function block.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the shutdown process is to be aborted. If it is to be run on the local computer, an empty string can be entered.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

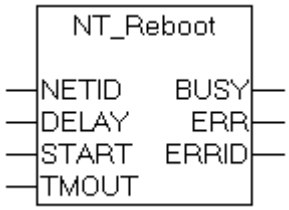
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.3 NT_Reboot



The Windows NT operating system can be restarted with the aid of the "NT_Reboot" function block. The function largely corresponds to the Restart command on the Windows taskbar. A delay before execution of the Restart command can be defined via the DELAY parameter. Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    DELAY       :DWORD;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer that is to be re-booted. If the restart is to take place on the local computer, an empty string can be entered.

DELAY: The delay time, in seconds, before the Restart command is executed.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

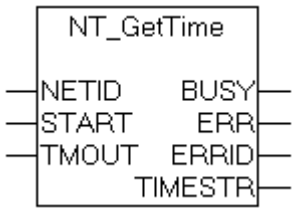
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.4 NT_GetTime



The local Windows system time of a TwinCAT system can be obtained using the "NT_GetTime" function block (the local Windows system time is displayed in the task bar). The year, month, day, day of the week, hour, minute, second an millisecond are placed in the variables in the structure TIMESTRUCT [▶ 230]. Internally, an instance of the ADSREAD function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer whose local Windows system time is to be read. If it is to be run on the local computer, an empty string can be entered.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

TIMESTR: Structure [▶ 230] with the local Windows system time.

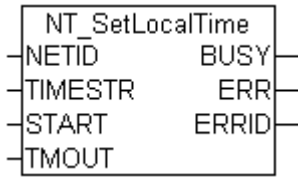
Further time, time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime, NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.5 NT_SetLocalTime



The function block "NT_SetLocalTime" sets the local Windows system time of the TwinCAT system (the local Windows system time is displayed in the task bar).

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TIMESTR     :TIMESTRUCT;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the local Windows system time is to be set. If it is to be run on the local computer, an empty string can be entered.

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

TIMESTR: Structure [▶ 230] with the new local Windows system time.

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

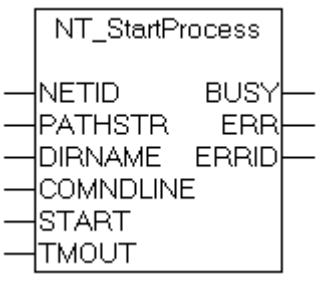
Further time, time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime, NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 and higher TwinCAT v2.10.0 Build > 1232 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.6 NT_StartProcess



The "NT_StartProcess" function block can be used to start a Windows application from the PLC. Internally, an instance of the ADSWRITE function block is called. The function block can also be used to run applications on a remote PC.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    PATHSTR     :T_MaxString;
    DIRNAME     :T_MaxString;
    COMNDLINE   :T_MaxString;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the application is to be started. If it is to be run on the local computer, an empty string can be entered.

PATHSTR: The full path to the application that is to be run, in the form of a string (e.g. "C:\WINNT\NOTEPAD.EXE").

DIRNAME: Working directory of the application that is to be executed, as a string (e.g. "C:\WINNT").

COMNDLINE: Command line parameters (e.g.: "win.ini").

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

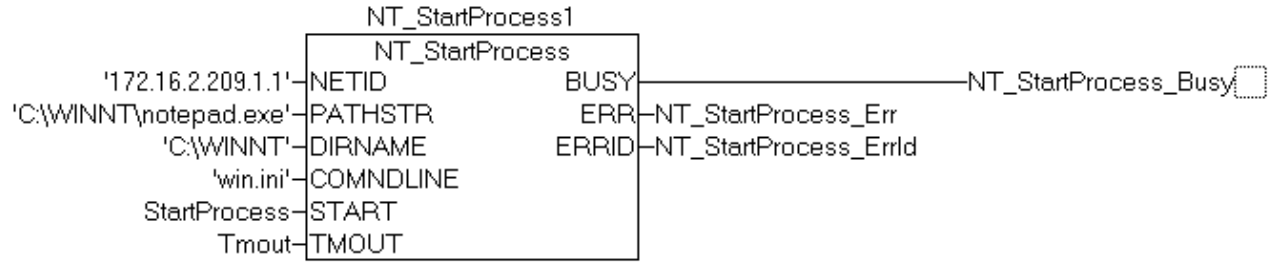
BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number or Win32 error number (Platform SDK: Win32 API) when the ERR output is set.

Example of a call in FBD

```
NT_StartProcess1        : NT_StartProcess;
NT_StartProcess_Busy    : BOOL;
NT_StartProcess_Err     : BOOL;
NT_StartProcess_ErrId   : UDINT;
StartProcess            : BOOL;
Tmout                   : TIME;
```

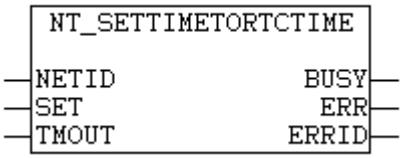


Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.7 NT_SetTimeToRTCTime

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



The function block "NT_SetTimeToRTCTime" can be used to synchronise the local Windows system time (displayed in the task bar) with the real-time clock of the PC (RTC time in the BIOS).

Remarks

When the function block is called the real-time clock of the TwinCAT PC is compared with the local Windows system time, and the local Windows system time is corrected based on the difference that was determined. Time zones and daylight-saving/standard time bias (difference) are considered. Please note that the correction can lead to time jumps during measurements or log book entries.

When the local Windows system time is set the operating system also automatically sets the RTC time to the new local Windows system time. Due to the conversion and delay the new RTC time is inevitably subject to a small error in the millisecond range, i.e. each time NT_SetTimeToRTCTime is called a small error is introduced in the real-time clock. In order to minimise deviations over a prolonged period, the compensation should be carried out every 24 hours, for example, rather than during each PLC cycle.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    SET         :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the local Windows system time is to be synchronisized. If it is to be run on the local computer, an empty

SET: The block is activated by a rising edge at this input.

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

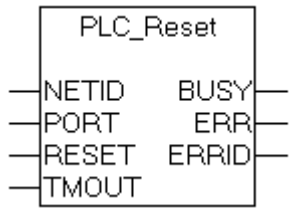
Further time, time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime, F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development Environment	Target System	PLC Libraries to include
TwinCAT v2.7.0 Build > 508	PC oder CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 Build > 715	PC oder CX (x86)	TcUtilities.Lib

4.8 PLC_Reset



The "PLC_Reset" function block can be used to reset a PLC run-time system. When the PLC is reset, the PLC variables are filled with their initial values, and the execution of the PLC program is stopped. Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

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

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the PLC run-time system is to be reset. If the PLC reset is to be carried out on the local computer, an empty string can be entered.

PORT: Contains the ADS port number of the PLC run-time system that is to be reset. The first PLC run-time system, for example, has port number 801, and the second has port number 811.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

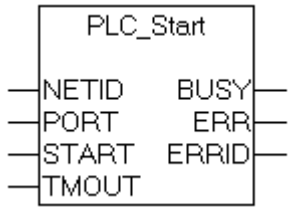
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.9 PLC_Start



The function block "PLC_Start" can be used to start a PLC run-time system on a TwinCAT computer. The function block can, for instance, be used to start the PLC on a remote PC.

Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

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

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the PLC is to be started. If the PLC is to be started on the local computer, an empty string can be entered.

PORT: Contains the ADS port number of the PLC run-time system that is to be started. The first PLC run-time system, for example, has port number 801, and the second has port number 811.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

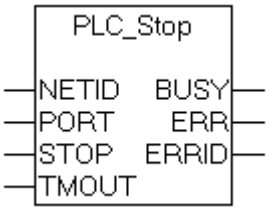
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.10 PLC_Stop



The function block "PLC_Stop" can be used to stop a PLC run-time system on a TwinCAT computer. The function block can, for instance, be used to stop the PLC on a remote or a local PC. Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

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

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the PLC is to be stopped. If the PLC to be stopped is on the local computer, an empty string can be entered.

PORT: Contains the ADS port number of the PLC run-time system that is to be stopped. The first PLC run-time system, for example, has port number 801, and the second has port number 811.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

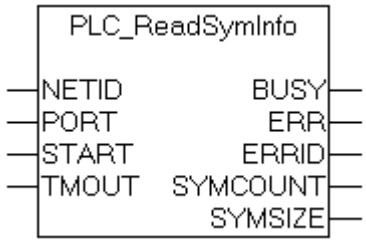
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.11 PLC_ReadSymInfo



The "PLC_ReadSymInfo" function block permits information regarding the symbols (variables) of a PLC run-time system. Internally, an instance of the ADSREAD function block is called.

VAR_INPUT

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

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer whose symbol information is to be found. If it is to be run on the local computer, an empty string can be entered.

PORT: The port number of a PLC run-time system.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

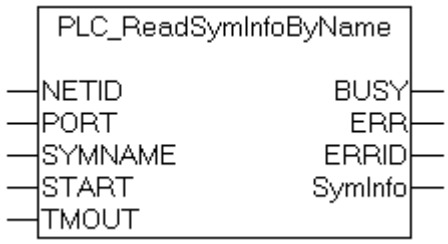
SYMCOUNT: The number of symbols in the PLC run-time system.

SYMSIZE: Length of the data, in bytes, in which the symbol information is stored.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.12 PLC_ReadSymInfoByName



The function block "PLC_ReadSymInfoByName" allows additional information about the PLC symbolic variables (e.g. data type identification, index group, index offset, comment...) to be read using the symbol name. After successful execution, the data is available in the **SymInfo** data structure, whose type is SYMINFOSTRUCT [▶ 232]. The symbolic variables are stored as individual entries in a symbol table. The size of each entry is variable and is not limited. It is necessary to provide a data buffer of the appropriate size to read an entry in the PLC so that it can be processed by the function block. Dynamic memory allocation in the PLC is not possible at run-time. In most cases a data buffer of 1000 bytes is sufficient. The symbol entry is copied into the data buffer during execution, decoded, and the required data is copied again into the **SymInfo** output variable. It is not possible to read or process the entry if the data buffer is too small. The size of the buffer depends heavily on the length of the symbol name and the length of the comment in the variable definition line. After successful execution it is possible to determine the buffer size that is in fact required from the **SymInfo.symEntrySize** output variable.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    PORT        :T_AmsPort;
    SYMNAME     :T_MaxString;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the function to be executed. If it is to be run on the local computer, an empty string can be entered.

PORT: The port number of the PLC run-time system to which the symbolic variable belongs. The first PLC run-time system has the port number 801.

SYMNAME: The symbol name of the PLC variable whose information is to be read (max. 255 characters, including the complete path, e.g. 'MAIN.INIT_TASK.VARINT').

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

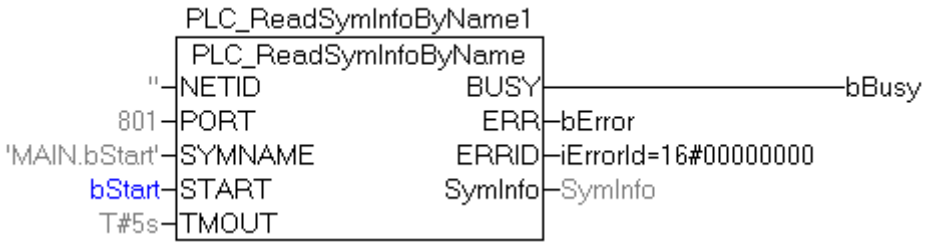
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

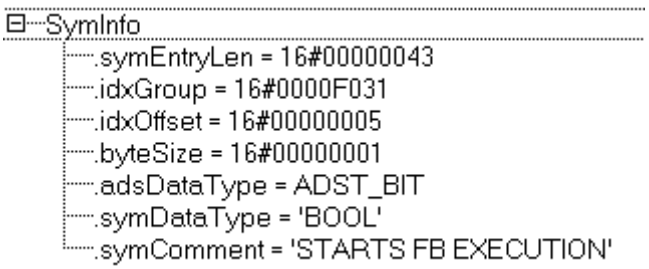
SymInfo: A structure [▶ 232] with additional information on the symbolic variable.

Example of a call in FBD

```
PROGRAM MAIN
VAR
    PLC_ReadSymInfoByName1      : PLC_ReadSymInfoByName;
    bStart  AT%QX0.5            : BOOL;     (*Starts FB execution*)

    tmpBuffer                   : ARRAY[1..76] OF BYTE;

    bBusy               : BOOL;
    bError              : BOOL;
    iErrorId                : UDINT;
    SymInfo             : SYMINFOSTRUCT;(*Structure with symbol information*)
END_VAR
```



Online View:



The data obtained in this way has the following meanings:

symEntryLen = 16#43: The actual length of the entry in the symbol table is 67 bytes;

idxGroup = *16#F031*: It is a variable from the PLC process image of the physical outputs;

idxOffset = *16#5*: The variable is located at byte offset zero and bit offset 5;

byteSize = *16#1*: The variable's value occupies one byte in memory;

adsDataType = *ADST_BIT*: The ADS data type ID;

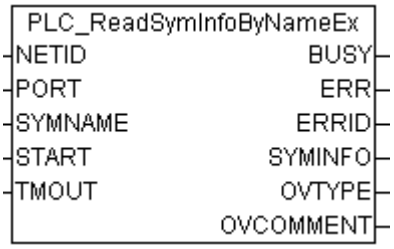
symDataType = *BOOL*: The data type identification in the PLC;

symComment = *'STARTS FB EXECUTION'*: Comment that the user has added to the variable definition line.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.13 PLC_ReadSymInfoByNameEx



The function block "PLC_ReadSymInfoByNameEx" has similar functionality as the function block PLC_ReadSymInfoByName [▶ 138]. Both function blocks can read symbol information through the symbol name. The difference between the two blocks is that the block described here does not report an error if the available buffer size is exceeded and may output incomplete information. In this case the comment and/or the data type ID may have been truncated. Two additional output variables indicate this, i.e.*OVTYPE* and *OVCOMMENT*, so that the application can respond accordingly.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    PORT        :T_AmsPort;
    SYMNAME     :T_MaxString;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the function to be executed. If it is to be run on the local computer, an empty string can be entered.

PORT: The port number of the PLC run-time system to which the symbolic variable belongs. The first PLC run-time system has the port number 801.

SYMNAME: The symbol name of the PLC variable whose information is to be read (max. 255 characters, including the complete path, e.g. 'MAIN.INIT_TASK.VARINT').

START: The block is activated by a rising edge at this input.

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
    SymInfo     : SYMINFOSTRUCT;
    OVTYPE      : BOOL;(* TRUE => Type name string length overflow, FALSE => no overflow *)
    OVCOMMENT   : BOOL;(* TRUE => Comment string length overflow, FALSE => no overflow *)
END_VAR
```

BUSY: When the function block is activated this output is set. It remains set until feedback is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

SymInfo: A structure [▶ 232] with additional information on the symbolic variable.

OVTYPE: Indicates whether the string with the data type identification has caused an overflow (TRUE). The string with the data type identification may have been truncated.

OVCOMMENT: Indicates whether the string with the symbol comment has caused an overflow (TRUE). The string with the comment may have been truncated.

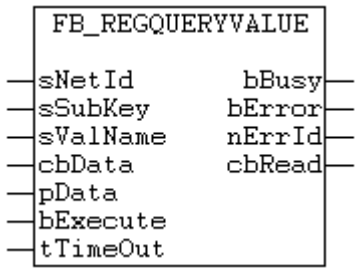
Example of a call in FBD

See function block documentation: PLC_ReadSymInfoByName [▶ 138]

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

4.14 FB_RegQueryValue



The System registration is a hierarchical structured tree. A node in the tree is called key. Each Key again can contain SubKeys, and Values.
With the function block "FB_RegQueryValue" single key values can be read from the system registry from the branch with the predefined handle **HKEY_LOCAL_MACHINE.** If this was successfull, the *cbData*- data bytes will be copied in the buffer with the address *pData* .

With the function block any value types (e.g. REG_DWORD, REG_SZ) or unlimited binary data (REG_BINARY) can be read.

Comment:

Empty *sSubKey* and *sValueName* strings are not allowed!

VAR_INPUT

```
VAR_INPUT
    sNetId      :T_AmsNetId;
    sSubKey     :T_MaxString;
    sValName    :T_MaxString;
    cbData      :UDINT;
    pData       :UDINT;
    bExecute    :BOOL;
    tTimeOut    :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: Here a string containing the network address of the TwinCAT Computer can be given, whose system registration should be read. The string can also be empty for the local computer.

sSubKey: String with SubKey name.

sValName: String with value name.

cbData: Number of value data bytes to be read.

pData: Address of a data buffer/variables to which the value data should be copied. The address can be ascertained from the ADR-Operator. The programmer himself is responsible for the dimension of the data buffer:.it must be possible to take *cbData* data bytes into the buffer.

bExecute: The command is executed with a positive edge at the input

tTimeOut: Maximum time allowed for the execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

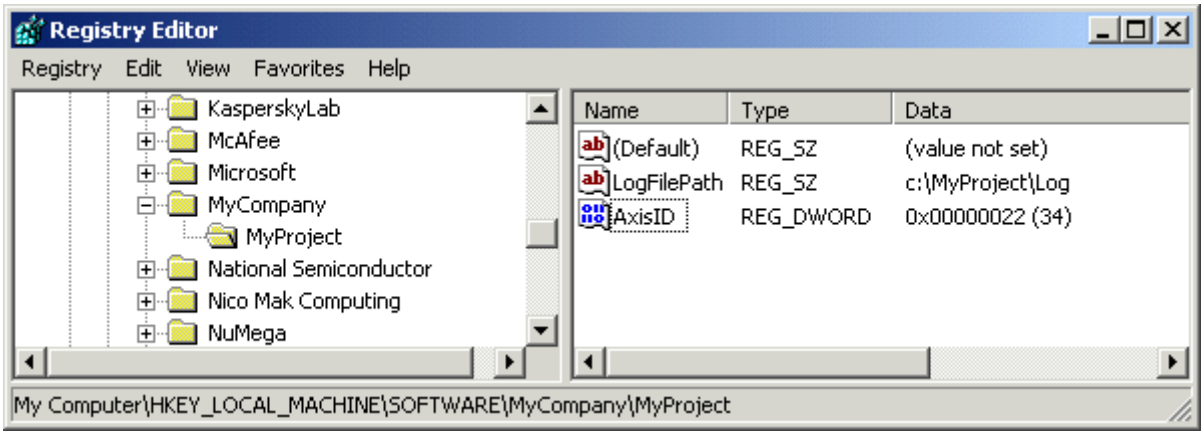
nErrId: When the bError output is set, this variable supplies the ADS error or the command-specific error code (table).

cbRead: Number of successfully read value data bytes.

Error codes	Error description
0x00	no error
0x01	The key with the name **sSubKey** could not be opened/found.
0x02	The key value with the name **sValName** could not be opened/found.

Examples:

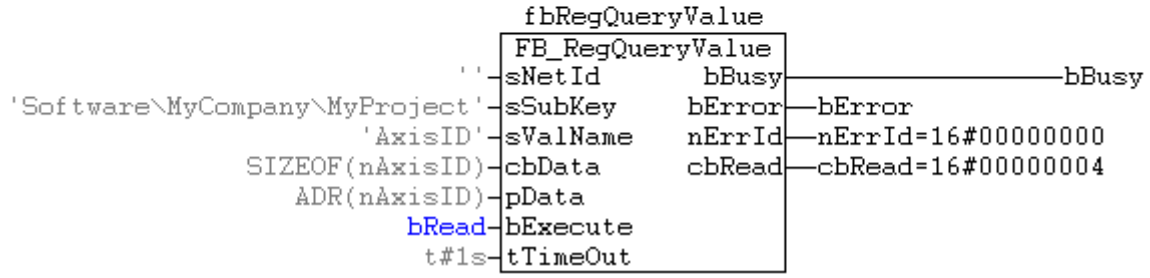
The values *AxisID* and *LogFilePath* should be read from the system Registry.




```
PROGRAM MAIN
VAR
    fbRegQueryValue : FB_RegQueryValue;
    bRead           : BOOL;
    bBusy           : BOOL;
    bError          : BOOL;
    nErrId          : UDINT;
    cbRead          : UDINT;
    sValData        : STRING;
    nAxisID         : DWORD;
END_VAR
```

Read REG_DWORD-Value:

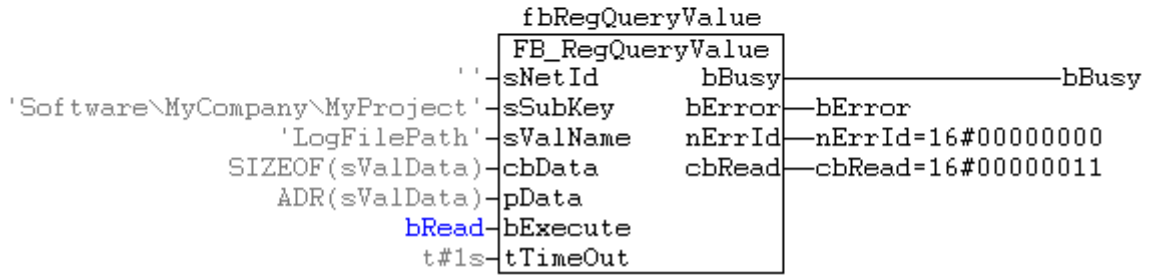
```
fbRegQueryValue
    bRead = TRUE
    bBusy = FALSE
    bError = FALSE
    nErrId = 16#00000000
    cbRead = 16#00000004
    nAxisID = 16#00000022
```



The value 0x22 of the registry was read in the PLC variable *nAxisId* .

Read REG_SZ-Value :

```
fbRegQueryValue
   bRead = TRUE
   bBusy = FALSE
   bError = FALSE
   nErrId = 16#00000000
   cbRead = 16#00000011
   sValData = 'c:\MyProject\Log'
```

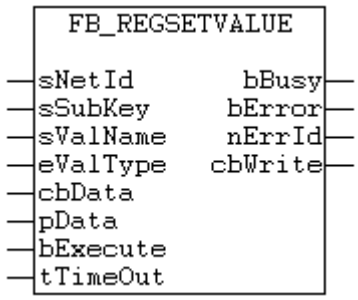


The string 'c:\MyProject\Log' of the registry was read in the PLC variable *sValData.*

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.7.0 Build > 519	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 Build > 723	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.15 FB_RegSetValue



The System registration is a hierarchical structured tree. A node in the tree is called key. Each Key again can contain SubKeys, and Values.
With the function block "FB_RegSetValue" single key values, or new SubKey names and Values can be written resp. generated in a branch with the predefined handle **HKEY_LOCAL_MACHINE.** Any value types (e.g. REG_DWORD, REG_SZ) or maximum 500 byte binary data (REG_BINARY) can be written in the system registry. If a key value doesn´t exist yet, a new one will be created automatically.

Comment:

Empty *sSubKey* and *sValueName* strings are not allowed!

VAR_INPUT

```
VAR_INPUT
    sNetId      :T_AmsNetId;
    sSubKey     :T_MaxString;
    sValName    :T_MaxString;
    eValType    :E_RegValueType;
    cbData      :UDINT;
    pData       :UDINT;
    bExecute    :BOOL;
    tTimeOut    :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: Here a string containing the network address of the TwinCAT Computer can be given, whose system registration should be written. The string can also be empty for the local computer.

sSubKey: String with SubKey name.

sValName: String with value name.

eValType: Data type format [▶ 233] of of the registration data to be written e.g. REG_DWORD or REG_SZ.

cbData: Number of value data bytes to be written (at string variables inclusive the terminal zero).

pD ata: Address of a data buffer/PLC variables, which contains the value data. The address can be ascertained from the ADR-Operator. The programmer himself is responsible for the dimension of the data buffer:.it must be possible to take *cbData* data bytes from the buffer.

bExecute: The command is executed with a positive edge at the input

tTimeOut: Maximum time allowed for the execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrId: When the bError output is set, this variable supplies the ADS error or the command-specific error code (table).

cbWrite: Number of successfully written value data bytes.

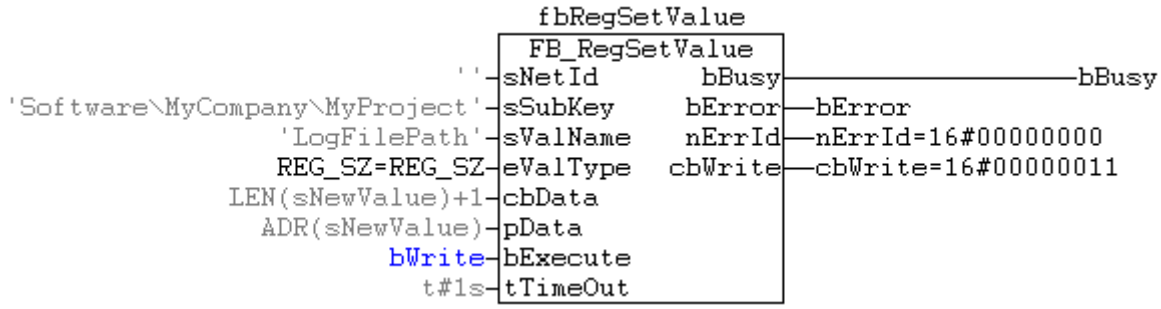
Error codes	Error description
0x00	no error
0x01	The key with the name **sSubKey** could not be opened/found.
0x02	The key value with the name **sValName** could not be opened/found.

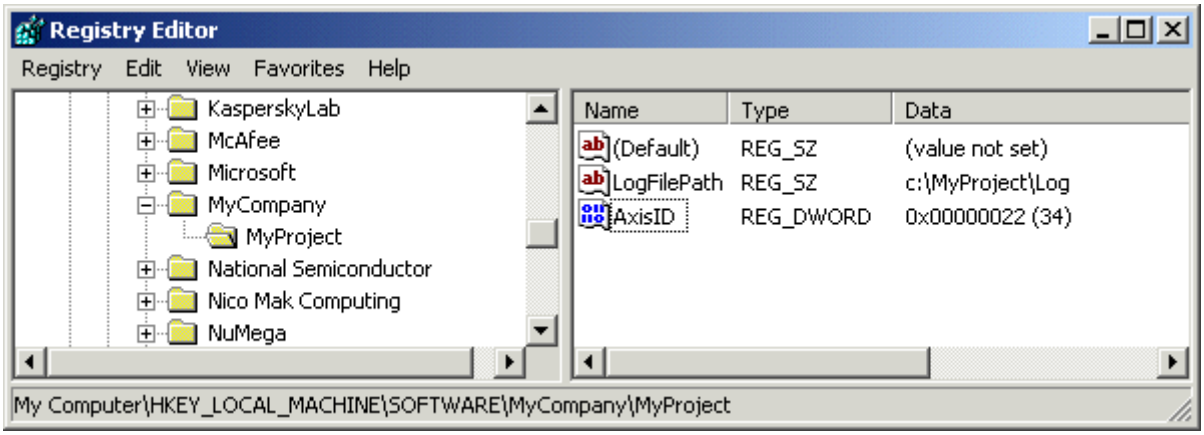
Example:

A Subkey *SOFTWARE\MyCompany\MyProject'* witt the key name '*LogFileName'* , the type *REG_SZ* and the value '*c:\MyProject\Log*" should be created and set in the branch with the predefined handle *HKEY_LOCAL_MACHINE.*

```
PROGRAM MAIN
VAR
    fbRegSetValue   : FB_RegSetValue;
    bBusy           : BOOL;
    bError          : BOOL;
    nErrId          : UDINT;
    cbWrite         : UDINT;
```

```
    bWrite          : BOOL;
    sNewValue               : STRING := 'c:\MyProject\Log';
END_VAR
```

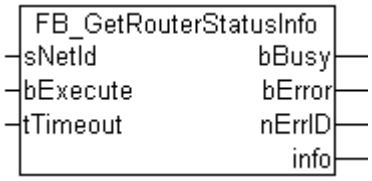




Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.7.0 Build > 519	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 Build > 723	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.16 FB_GetRouterStatusInfo



The function block FB_GetRouterStatusInfo reads the TwinCAT Router status information (available memory, number of registered ports...).

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetID := '';
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: Here a string containing the network address of the TwinCAT Computer can be given, whose TwinCAT Router status information should be read. The string can also be empty for the local computer.

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

tTimeOut: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the *bBusy* output is reset.

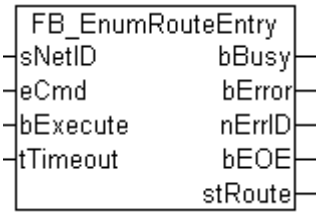
nErrId: Supplies the ADS error code when the *bError* output is set.

info: Struct [▶ 243] variable with TwinCAT Router status info.

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1235	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.17 FB_EnumRouteEntry



This function block is used to transfer information to other TwinCAT systems via the AMS router connections (remote routes). If several connections are used the function block must be called up repeatedly. Only one entry can be handled for each call. The input parameter *eCmd* is used for navigating through the list of entries. The *eCmd* input determines whether the first or the next input is read.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the network address of the TwinCAT Computer can be given, whose AMS Router connection should be read. The string can also be empty for the local computer.

eCmd: Control command [▶ 235] for the enumeration block.

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    bEOE    : BOOL;
    stRoute : ST_AmsRouteEntry;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrId: When the bError output is set, this variable supplies the ADS error code.

bEOE: End of enumeration was reached. During the first attempt to read a non-existing entry this output is set to TRUE. This means that read entries are valid if *bEOE* = FALSE and *bError* = FALSE.

stRoute: Structure [▶ 243] element with latest read connection parameter.

Example in ST:

The configured AMS router connections are to be read on the local TwinCAT system and written into the TwinCAT System Manager logger output as messages.

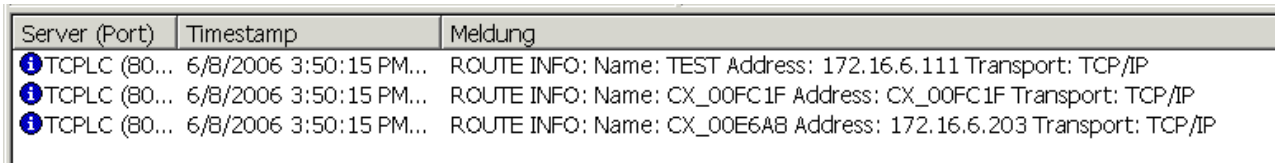
```
PROGRAM P_EnumRouteEntries
VAR
    fbEnum  : FB_EnumRouteEntry := ( sNetID := '', tTimeout := T#5s );
    bEnum   : BOOL := TRUE;
    nState  : BYTE := 0;
    sInfo   : T_MaxString;
END_VAR
```

A rising edge at the bEnum variable triggers reading of the connection information.

```
CASE nState OF
    0:
        IF bEnum THEN (* flag set ? *)
         bEnum   := FALSE;           (* reset flag *)
         fbEnum.eCmd := eEnumCmd_First;  (* enum first entry *)
         nState  := 1;
        END_IF

    1:      (* enum one entry *)
     fbEnum( bExecute := FALSE );
        fbEnum( bExecute := TRUE );
        nState  := 2;

    2:      (* wait until function block not busy *)
     fbEnum( bExecute := FALSE );
        IF NOT fbEnum.bBusy THEN
            IF NOT fbEnum.bError THEN
                IF NOT fbEnum.bEOE THEN
                    sInfo := CONCAT( 'Name: ', fbEnum.stRoute.sName );
                    sInfo := CONCAT( sInfo, ' Address: ' );
                    sInfo := CONCAT( sInfo, fbEnum.stRoute.sAddress );
                    sInfo := CONCAT( sInfo, ' Transport: ' );
                    sInfo := CONCAT( sInfo, ROUTETRANSPORT_TO_STRING( fbEnum.stRoute.eTransport ) );
                    ADSLOGSTR( ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_LOG, 'ROUTE INFO: %s', sInfo );
                    fbEnum.eCmd := eEnumCmd_Next; (* enum next entry *)
                 nState  := 1;
                ELSE (* no more route entries *)
                 nState  := 0;
                END_IF
            ELSE (* log error *)
             ADSLOGSTR( ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_LOG, 'FB_EnumRouteEntry error:
%s', DWORD_TO_HEXSTR( fbEnum.nErrID, 0, FALSE ) );
                nState  := 0;
```

```
            END_IF
        END_IF
END_CASE
```

The written Log messages in the TwinCAT System Manager Logger output:

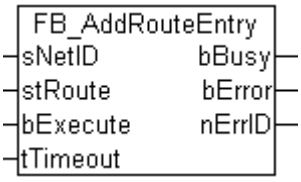


Server (Port)	Timestamp	Meldung
TCPLC (80...	6/8/2006 3:50:15 PM...	ROUTE INFO: Name: TEST Address: 172.16.6.111 Transport: TCP/IP
TCPLC (80...	6/8/2006 3:50:15 PM...	ROUTE INFO: Name: CX_00FC1F Address: CX_00FC1F Transport: TCP/IP
TCPLC (80...	6/8/2006 3:50:15 PM...	ROUTE INFO: Name: CX_00E6A8 Address: 172.16.6.203 Transport: TCP/IP

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.18 FB_AddRouteEntry



This function block adds a new AMS Router connection (remote route) to a TwinCAT System.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    stRoute     : ST_AmsRouteEntry;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the network address of the TwinCAT Computer can be given, whose AMS Router connection list should be added by a new connection. The string can also be empty for the local computer.

stRoute: Structure [▶ 243] element with parameters of the new connection.

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

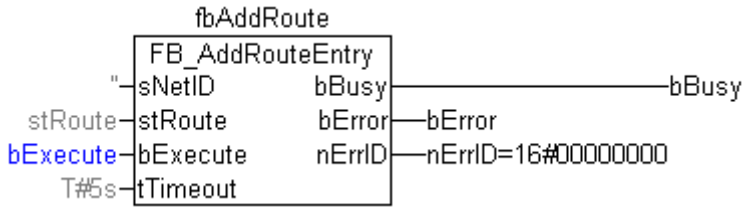
nErrId: When the bError output is set, this variable supplies the ADS error code.

Example in FBD:

A new AMS Router connection should be added on the local TwinCAT System with the connection name: "TEST", TwinCAT network address: "172.16.6.111.1.1", IP address: "172.16.6.111" and the transport medium: "TCP/IP".

```
PROGRAM P_TEST3
VAR
    fbAddRoute  : FB_AddRouteEntry;
    bExecute    : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;

    stRoute : ST_AmsRouteEntry := (     sName := 'TEST',
                    sNetID := '172.16.6.111.1.1',
                    sAddress := '172.16.6.111',
                    eTransport := eRouteTransport_TCP_IP );
END_VAR
```

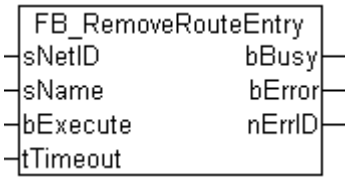
The required connection parameters are initialised in the declaratoin part. The new connection is added by a positive edge a the bExecute variable.



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.19 FB_RemoveRouteEntry



This function block deletes an existing connection to a TwinCAT System from the AMS Router Connection (remote routes) list.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    sName       : STRING(MAX_ROUTE_NAME_LEN);
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the network address of the TwinCAT Computer can be given, whose AMS Router connection should be deleted. The string can also be empty for the local computer.

sName: Connection name of the connection to be deleted. The maximum string length is limited by a constant (default: 31 characters).

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

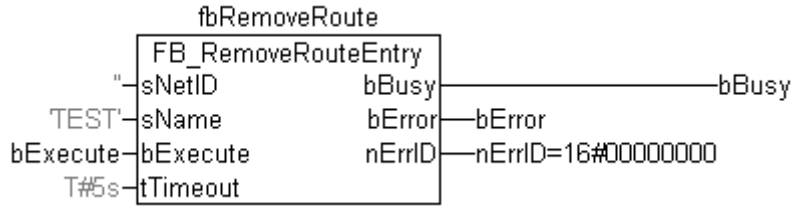
bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrId: When the bError output is set, this variable supplies the ADS error or the command-specific error code.

Example in FBD:

The connection with the connection name: "TEST" should be deleted from the list of the AMS Router connections on the local TwinCAT System. The connection is deleted by a positive edge a the bExecute variable.

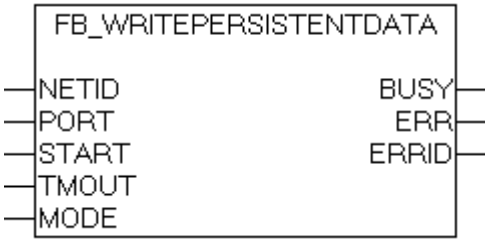
```
PROGRAM P_TEST2
VAR
    fbRemoveRoute   : FB_RemoveRouteEntry;
    bExecute    : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
END_VAR
```



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.20 FB_WritePersistentData



The function block FB_WritePersistentData is an extended version of WritePersistentData [▶ 216] function block. The input parameter MODE affects the system behaviour of how the persistent data is written. For additional info see: Writing of persistent data: System behaviour [▶ 277].

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : UINT;
    START       : BOOL;
    TMOUT       : TIME := DEFAULT_ADS_TIMEOUT;
    MODE        : E_PersistentMode;
END_VAR
```

NETID : It is possible here to provide the AmsNetId of the TwinCAT computer on which the ADS command is to be executed. If it is to be run on the local computer, an empty string can be entered.

PORT : Contains the ADS port number of the PLC run-time system whose persistent data is to be stored. The first PLC run-time system, for example, has port number 801, and the second has port number 811.

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

TMOUT : States the length of the timeout that may not be exceeded by execution of the ADS command.

MODE : Persistent data write mode [▶ 235]. For additional info see: Writing of persistent data: System behavior [▶ 277].

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

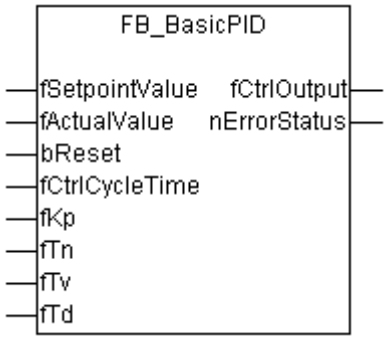
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >= 959	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.21 FB_BasicPID

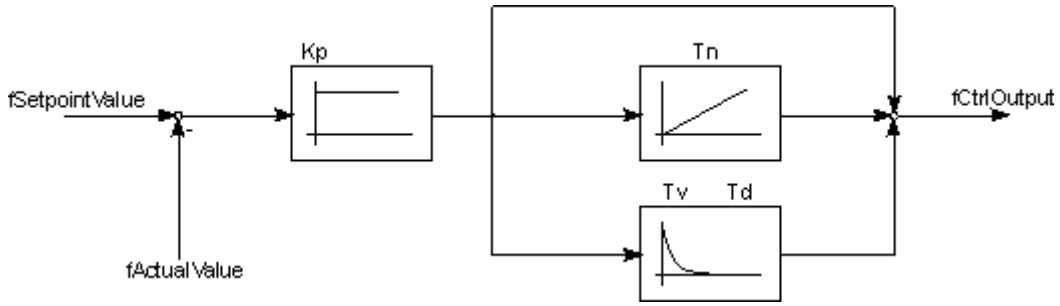


The function block represents a simple discrete PID element.

Transfer function:

$$G(s) = K_p(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s})$$

Action diagram:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : LREAL; (* setpoint value *)
    fActualValue    : LREAL; (* actual value *)
    bReset          : BOOL;
    fCtrlCycleTime  : LREAL; (* controller cycle time in seconds [s] *)
    fKp             : LREAL; (* proportional gain Kp (P) *)
    fTn             : LREAL; (* integral gain Tn (I) [s] *)
    fTv             : LREAL; (* derivative gain Tv (D-T1) [s] *)
    fTd             : LREAL; (* derivative damping time Td (D-T1) [s] *)
END_VAR
```

fSetpointValue : Setpoint value of the controlled variable.

fActualValue : Actual value of the controlled variable.

bReset : TRUE at this input resets the internal state variables as well as the output of the controller.

fCtrlCycleTime : Controller cycle time for discrete-time implementation [s].

It is **required** to set this value equal to the task cycle time of the PLC task, if the function block is called every PLC cycle.

Otherwise the appropriate multiple of the PLC task cycle time has to be used.

fKp : Proportional amplification

fTn : Integral-action time [s]

fTv : Derivative action time [s]

fTd : Damping time [s]

VAR_OUTPUT

```
VAR_OUTPUT
    fCtrlOutput     : LREAL;
    nErrorStatus    : UINT
END_VAR
```

fCtrlOutput : output of thePID element.

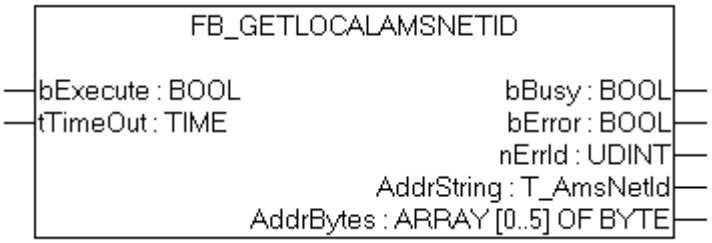
nErrorStatus : Provides the error number, if an error exists (nErrorStatus <> 0).

0 = nERR_NOERROR : no error.
1 = nERR_INVALIDPARAM : invalid parameter
2 = nERR_INVALIDCYCLETIME : Invalid cycle time

Requirements

Development Environment	Target System	PLC Libraries to include
TwinCAT v2.7.0 Build > 519	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 Build > 739	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

4.22 FB_GetLocalAmsNetId



The function block FB_GetLocalAmsNetId reads the local AmsNetId (local TwinCAT-specific network address).

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    tTimeOut        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

bExecute: The command is executed with a rising edge at the input

tTimeOut: Maximum time allowed for the execution of this ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    nErrId      :UDINT;
    AddrString  :T_AmsNetId;
    AddrBytes   :T_AmsNetIdArr;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until an acknowledgement is received.

bError: If an ADS error occurs during execution, this output is set after *bBusy* output has been reset.

nErrId: This variable provides the ADS error code, if *bError* is true.

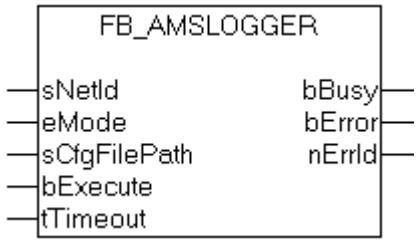
AddrString: AmsNetId as string.

AddrBytes: AmsNetId as array of byte.

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.8.0 Build > 744 TwinCAT v2.9.0 Build > 941	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.23 FB_AmsLogger



The "TwinCAT AMS Logger" belongs to the components of the "TwinCAT ADS Monitor" (.. \TwinCAT\AdsMonitor\Logger**TcAmsLog.exe**). The logger charts the AMS/ADS commands to a data medium. The chart can for e.g. be displayed by "TwinCAT AMS ADS Viewer" and analysed for debugging. The record of the "TwinCAT AMS Logger" can be started or stopped from the PLC program using the instance of function block "FB_AmsLogger". The instance of TcAmsLog.exe have allready be existing and running. Please use start menue command or e.g. NT_StartProcess [▶ 131] function block to start an instance of TcAmsLog.exe.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId:= '';
    eMode       : E_AmsLoggerMode := AMSLOGGER_RUN;
    sCfgFilePath: T_MaxString := '';
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: It is possible here to provide the network address of the TwinCAT computer on which the "TwinCAT AMS Logger" state is to be changed. If the logger state is to be changed on the local computer, an empty string can be entered.

eMode: New "TwinCAT AMS Logger" state [▶ 234] to set.

sCfgFilePath: (Optionally) Path of the "TwinCAT AMS Logger" configuration file. Currently not used and reserved for future use (enter empty string).

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

bBuse: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

nErrId: Supplies the ADS error number when the ERR output is set.

Example:

The sample program starts new instance of TcAmsLog.exe and the record chart. Setting bRecord to FALSE stopps the recording.

Declaration part:

```
PROGRAM MAIN
VAR
    bRecord     : BOOL := TRUE;(* TRUE => start recording, FALSE => stop recording *)

    fbStartProcess  : NT_StartProcess := ( NETID := '', PATHSTR:= 'c:
\TwinCAT\AdsMonitor\Logger\TcAmsLog.exe',
                    DIRNAME:= 'c:
\TwinCAT\AdsMonitor\Logger', COMNDLINE:= '', TMOUT := DEFAULT_ADS_TIMEOUT );

    fbAmsLogger     : FB_AmsLogger := ( sNetID := '', eMode := AMSLOGGER_STOP, sCfgFilePath := '', t
Timeout := DEFAULT_ADS_TIMEOUT );
    state       : BYTE;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    eCurrMode   : E_AmsLoggerMode := AMSLOGGER_STOP;(* Current mode/state *)
    eNewMode    : E_AmsLoggerMode := AMSLOGGER_STOP;(* New mode/state *)
    timer       : TON := ( PT := T#5s );
END_VAR
```

Implementation:

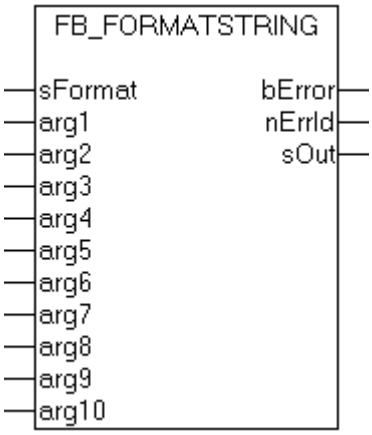
```
CASE state OF
0:(* Start instance of TcAmsLogger.exe *)
    fbStartProcess( START := FALSE );
    fbStartProcess( START:= TRUE );
    state := 1;

1:(* Wait until command execution started *)
    fbStartProcess( START := FALSE, BUSY=>bBusy, ERR=>bError, ERRID=>nErrID );
    IF NOT bBusy THEN
        IF NOT bError THEN(* Success *)
            state := 2;
        ELSE(* Error *)
            state := 100;
        END_IF
    END_IF

2:(*Wait until instance started or new AMS logger mode/state set *)
    timer( IN := TRUE );
    IF timer.Q THEN
        timer( IN := FALSE );
        state := 3;
    END_IF

3:(* Change TcAmsLog.exe mode/state *)
    eNewMode := SEL( bRecord, AMSLOGGER_STOP, AMSLOGGER_RUN);
```

```
    IF ( eNewMode <> eCurrMode ) THEN
        fbAmsLogger( bExecute := FALSE );
        fbAmsLogger( eMode:= eNewMode, bExecute := TRUE );
        state := 4;
    END_IF

4:(* Wait until command execution started *)
 fbAmsLogger( bExecute := FALSE, bBusy=>bBusy, bError=>bError, nErrID=>nErrID );
    IF NOT bBusy THEN
        IF NOT bError THEN(* Success *)
            eCurrMode := eNewMode;
            state := 2;
        ELSE(* Error *)
            state := 100;
        END_IF
    END_IF

100:(* Error state *)
    ;
END_CASE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build >= 959	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.24 FB_FormatString



This function block can be used for converting up to 10 arguments (similar to fprintf) into a string and formatting them according to the format specification [▶ 275]. Formatting occurs in the same PLC cycle, i.e. the output string is available immediately after the FB was called.

VAR_INPUT

```
VAR_INPUT
    sFormat : T_MaxString;
    arg1    : T_Arg;
    arg2    : T_Arg;
    arg3    : T_Arg;
    arg4    : T_Arg;
    arg5    : T_Arg;
    arg6    : T_Arg;
    arg7    : T_Arg;
    arg8    : T_Arg;
    arg9    : T_Arg;
    arg10   : T_Arg;
END_VAR
```

sFormat: format specification as string (e.g. '%+20.5f' or 'Measure X: %+.10d, Y: %+.10d').

arg1 bis **arg10**: arguments to be formatted. The following auxiliary functions can be used for converting different types of PLC variables into the required data type T_Arg [▶ 239]: F_BYTE [▶ 105], F_WORD [▶ 105], F_DWORD [▶ 106], F_SINT [▶ 106], F_INT [▶ 107], F_DINT [▶ 107], F_USINT [▶ 108], F_UINT [▶ 108], F_UDINT [▶ 109], F_STRING [▶ 109], F_REAL [▶ 104], F_LREAL [▶ 104].

VAR_OUTPUT

```
VAR_OUTPUT
    bError  : BOOL;
    nErrId  : UDINT;
    sOut    : T_MaxString;
END_VAR
```

bError: This output is set if an error occurs during formatting.

nErrId: Returns the format error code [▶ 250] if the bError output is set.

sOut: If successful, this output returns the formatted output string.

Example in ST:

```
PROGRAM MAIN
VAR
    fbFormat    : FB_FormatString;
    iY      : DINT;
    iX      : DINT;
    bError  : BOOL;
    nErrID  : UDINT;
    sOut    : T_MaxString;
END_VAR
```

```
iX := iX + 1;
iY := iY + 1;
fbFormat( sFormat := 'Measure X: %+.10d, Y: %
+.10d', arg1 := F_DINT( iX ), arg2 := F_DINT( iY ), sOut => sOut, bError => bError, nErrID => nErrID
 );
```

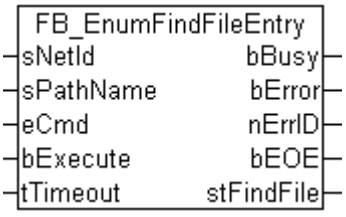
The result:

sOut = 'Measure X: +0000000130, Y: +0000000130'

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.25 FB_EnumFindFileEntry



This function block searches a directory for a file or a subdirectory whose name is similar to the specified name. Any entries found can be read individually. See also description of the FB_EnumFindFileList [▶ 160] function block. The input parameter *eCmd* is used for navigating through the list of entries. The *eCmd* input determines whether the first or the next input is read, for example.

Background information

A new search may be started only if the previous search has been fully completed. The function block may need to be activated several times (by a rising edge at the bExecute input) for a complete search. The search is only fully complete if bEOE =TRUE was reached or if the search was terminated prematurely with ECMD = eEnumCmd_Abort.

For the TwinCAT system, the search may not yet be completed if the PLC application has already found the file or directory that was sought.

If not all entries are to be read (i.e. *bEOE=TRUE* is not reached), the function block subsequently has to be called with the input parameter *eCmd = eEnumCmd_Abort*. This is necessary in order to complete the search and release all internal resources (file handles). If *bEOE=TRUE* was reached or if an error occurs, *eEnumCmd_Abort* is automatically executed internally.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    sPathName   : T_MaxString;
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: String containing the network address of the TwinCAT computer on which a directory search is to be executed. If it is to be run on the local computer, an empty string can be entered.

sPathName: Valid directory name or directory with file name as string. The string can contain (* and ?) as wildcards. If the path ends with a wildcard, dot or the directory name, the user must have access rights to this path and its subdirectories.

eCmd: Command parameter [▶ 235] for the enumeration block.

bExecute: The block is activated by a rising edge at this input.

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    bEOE        : BOOL;
    stFindFile  : ST_FindFileEntry;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until a feedback is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the *bBusy* output is reset.

nErrId: Supplies the ADS error number when the *bError* output is set.

bEOE: End of enumeration was reached. During the first attempt to read a non-existing entry this output is set to TRUE. This means that read entries are valid if *bEOE* = FALSE and *bError* = FALSE.

stFindFile: If successful this structure [▶ 244] variable returns information about the file that was found.

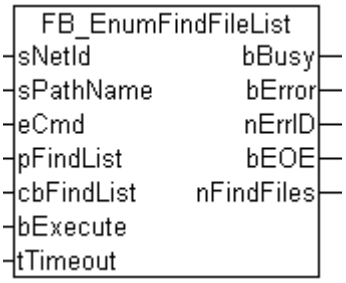
Example:

See: Example: File search (FB_EnumFindFileEntry, FB_EnumFindFileList). [▶ 255]

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build > 1302	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.26 FB_EnumFindFileList



This function block searches a directory for a file or a subdirectory whose name is similar to the specified name. Any entries found can be read individually. See also description of the FB_EnumFindFileEntry [▶ 158] function block. The input parameter *eCmd* is used for navigating through the list of entries. The *eCmd* input determines whether the first or the next input is read, for example.

Background information

A new search may be started only if the previous search has been fully completed. The function block may need to be activated several times (by a rising edge at the bExecute input) for a complete search. The search is only fully complete if bEOE =TRUE was reached or if the search was terminated prematurely with ECMD = eEnumCmd_Abort.

For the TwinCAT system, the search may not yet be completed if the PLC application has already found the file or directory that was sought.

If not all entries are to be read (i.e. *bEOE=TRUE* is not reached), the function block subsequently has to be called with the input parameter *eCmd = eEnumCmd_Abort*. This is necessary in order to complete the search and release all internal resources (file handles). If *bEOE=TRUE* was reached or if an error occurs, *eEnumCmd_Abort* is automatically executed internally.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    sPathName   : T_MaxString;
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pFindList   : DWORD;
    cbFindList  : UDINT;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: Here a string containing the network address of the TwinCAT Computer can be given, whose system registration should be read. The string can also be empty for the local computer.

sPathName: Valid directory name or directory with file name as string. The string can contain (* and ?) as wildcards. If the path ends with a wildcard, dot or the directory name, the user must have access rights to this path and its subdirectories.

eCmd: Command parameter [▶ 235] for the enumeration block.

pFindList: Array variable address (pointer variable) of type: ST_FindFileEntry.

cbFindList: Array variable byte size of type: ST_FindFileEntry.

bExecute: The command is executed with a positive edge at the input

tTimeOut: Maximum time allowed for the execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrId: When the bError output is set, this variable supplies the ADS error code.

bEOE: End of enumeration was reached. During the first attempt to read a non-existing entry this output is set to TRUE. This means that read entries are valid if *bEOE* = FALSE and *bError* = FALSE

nFindFiles: Number of valid files in the buffer.

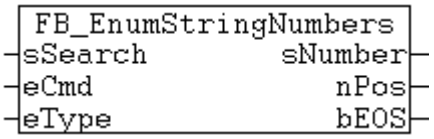
Example:

See: Example: File search (FB_EnumFindFileEntry, FB_EnumFindFileList). [▶ 255]

Requirements

Development Environment	Target System	PLC LIbraries to include
TwinCAT v2.10.0 Build > 1302	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.27 FB_EnumStringNumbers



This function block can be used to search a string in a REPEAT or WHILE loop for numbers. The string may contain several numbers. Any numbers that are found are output as sub-strings at the block output. The function searches from the current position for the first character that can be interpreted as a numeral. The search is aborted if a character is found that cannot be interpreted as a number. The *eCmd* parameter determines whether the first or the next number is read. The *eType* parameter determines the number format in the search string.

VAR_INPUT

```
VAR_INPUT
    sSearch : T_MaxString;
    eCmd    : E_EnumCmdType := eEnumCmd_First;
    eType   : E_NumGroupTypes := eNumGroup_Float;
END_VAR
```

sSearch: Search string to be searched for numbers.

eCmd: Command parameter [▶ 235] for the enumeration block.

eType: Type [▶ 236] of the searched number. This parameter determines which characters are to be ignored and which are to be interpreted as numerals:

Value	Description
eNumGroup_Float	Numbers '0' to '9', '+', '-' (signs) and 'e' or 'E' (exponent character) are interpreted as valid numerals.
eNumGroup_Unsigned	Numbers '0' to '9' are interpreted as valid numerals. '+', '-', 'e', 'E' characters are ignored.
eNumGroup_Signed	Numbers '0' to '9', '+', '-' (signs) are interpreted as valid numerals. 'e', 'E' characters are ignored.

If the string contains numbers in the exponential notation, eType = *eNumGroup_Float* must be set (default).

VAR_OUTPUT

```
VAR_OUTPUT
    sNumber : T_MaxString;
    nPos    : INT;
    bEOS    : BOOL;
END_VAR
```

sNumber: The last found number as string.

nPos: This variable always returns the position after the last found and correctly formatted numeral, i.e. at this position the block will start searching for new numerals when it is called next time. *nPos* is zero when the final zero of the *sSearch* string has been reached. The first character in the string has position number = 1 (non-zero based position).

bEOS: This variable is FALSE if a new number was found and the end of the string has not yet been reached. In this case *sNumber* returns a valid number as a string. This variable is TRUE if no further number was found. In this case any further search must be aborted (*sNumber* returns no valid value).

Example in ST:

In the following example the *sNumber* variable is searched for valid numbers. Any sub-strings that are found are stored in the array variable *arrNums*.

```
TYPE ST_ScanRes :
STRUCT
    sNumber     : T_MaxString;
    nPos    : INT;
    sRemain     : T_MaxString;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    sSearch     : T_MaxString := 'Some numbers in string: +-12e-34, -56, +78';
    fbEnum  : FB_EnumStringNumbers := ( eType := eNumGroup_Float (* eNumGroup_Signed, eNumGroup_Unsi
gned *) );
    arrNums     : ARRAY[1..MAX_SCAN_NUMS] OF ST_ScanRes;
    idx     : INT;
    length  : INT;
    bEnum   : BOOL := TRUE;
END_VAR
VAR CONSTANT
    MAX_SCAN_NUMS : INT := 10;
END_VAR
```

```
IF bEnum THEN
    bEnum := FALSE;

    MEMSET( ADR( arrNums ), 0, SIZEOF( arrNums ) );
    idx := 0;
    length := LEN( sSearch );

    fbEnum( sSearch := sSearch, eCmd := eEnumCmd_First );
    WHILE NOT fbEnum.bEOS DO
        IF idx < MAX_SCAN_NUMS THEN
            idx := idx + 1;

            arrNums[idx].sNumber:= fbEnum.sNumber;
            arrNums[idx].nPos := fbEnum.nPos;
            IF fbEnum.nPos <> 0 THEN
                arrNums[idx].sRemain:= RIGHT( sSearch, length - fbEnum.nPos + 1 );
            END_IF
```

```
        END_IF
        fbEnum( eCmd := eEnumCmd_Next );
    END_WHILE
END_IF
```

Found strings:

- '-12e-34'
- '-56'
- '+78'

eType-Parameter *eNumGroup_Signed* supplies the following results:

- '-12'
- '-34'
- '-56'
- '+78'

eType-Parameter *eNumGroup_Unsigned* supplies the following results:

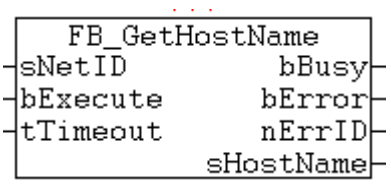
- '12'
- '34'
- '56'
- '78'

Requirements

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

4.28 FB_GetHostName

This functionality is supported only under Windows CE!



This function block reads the local host name of the TwinCAT computer.

VAR_INPUT

```
VAR_INPUT
    sNetID   : T_AmsNetId;
    bExecute : BOOL;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the TwinCAT network address of the TwinCAT computer can be given, whose local host name should be read. The string can also be empty for the local computer.

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the command.

VAR_OUTPUT

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

```
    nErrID    : UDINT;
    sHostName : T_MaxString;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

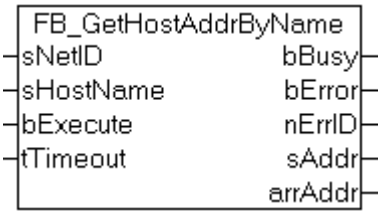
nErrID: When the *bError* output is set, this variable supplies the ADS error code.

sHostName: Local host name as string.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.29 FB_GetHostAddrByName



The function block reads hosts (IPv4) Internet Protocol network address corresponding to a host name from a host database. The address is returned as string and byte array.

VAR_INPUT

```
VAR_INPUT
    sNetID    : T_AmsNetId;
    sHostName : T_MaxString := '';
    bExecute  : BOOL;
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the TwinCAT network address of the TwinCAT computer can be given, where the command have to be executed. The string can also be empty (default) for the local computer.

sHostName: String containing host name. E.g. 'DataServer1'.

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the command.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrID  : UDINT;
    sAddr   : T_IPv4Addr := '';
    arrAddr : T_IPv4AddrArr := 0, 0, 0, 0;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrID: When the *bError* output is set, this variable supplies the ADS error code.

sAddr: String containing an (Ipv4) Internet Protocol dotted address. E.g. '172.16.7.199'

arrAddr: Byte array containing an (Ipv4) Internet Protocol dotted address.

Example in strutured text:

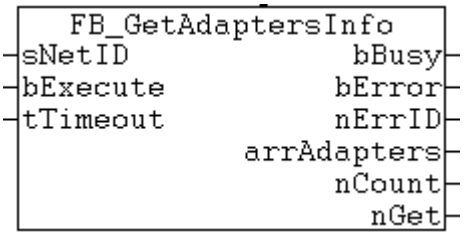
```
PROGRAM MAIN
VAR
    fbGet    : FB_GetHostAddrByName;
    bGet     : BOOL := TRUE;
    bError   : BOOL;
    nErrID   : UDINT;
    sIPv4    : T_IPv4Addr;(* Result: '87.106.8.100' *)
 arrIPv4     : T_IPv4AddrArr;
    state    : BYTE;
END_VAR
```

```
CASE state OF
0:
    IF bGet THEN
        bGet := FALSE;
        sIPv4 := '';
        fbGet( bExecute:= FALSE );
        fbGet( bExecute:= TRUE, sHostName := 'www.beckhoff.com' );
        state := 1;
    END_IF
1:
    fbGet( bExecute:= FALSE, bError=>bError, nErrID=>nErrID, sAddr=>sIPv4, arrAddr=>arrIPv4 );
    IF NOT fbGet.bBusy THEN
        state := 0;
    END_IF
END_CASE
```

Requirements

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

4.30 FB_GetAdaptersInfo



The function block retrieves adapter information for the local or remote TwinCAT computer. The max. number of read adapter information is limited to MAX_LOCAL_ADAPTERS + 1 (default = 6).

VAR_INPUT

```
VAR_INPUT
    sNetID   : T_AmsNetId;
    bExecute : BOOL;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: It is possible here to provide the network address of the TwinCAT computer on which the local adapter info should be read. The string can also be empty for the local computer.

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

tTimeout: States the length of the timeout that may not be exceeded by execution of the command.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    arrAdapters : ARRAY[0..MAX_LOCAL_ADAPTERS] OF ST_IpAdapterInfo;
    nCount      : UDINT;
    nGet        : UDINT;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrID: When the *bError* output is set, this variable supplies the ADS error code.

arrAdapters: Array variable with adapter information [▶ 245].

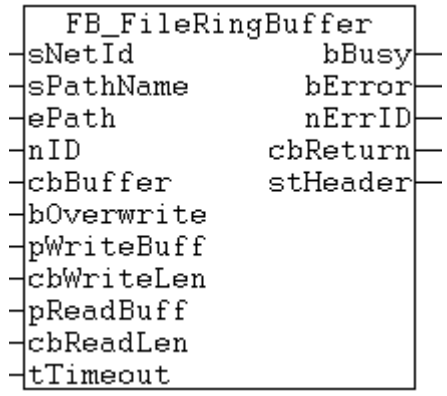
nCount: Max. number of found adapters.

nGet: Number of read adapter info entries.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.31 FB_FileRingBuffer



The FB_FileRingBuffer function block allows data records of varying lengths to be written into a ring buffer file, or for data records that have previously been written there to be removed from the ring buffer file. The file operates as a FIFO; data records are read from it in the same sequence in which they were first written into the ring buffer file. This means that the oldest entries are the first ones that are read. Opening, closing, writing, and reading the data records is controlled by action calls. The function block features the following tasks:

- **A_Open** (Opens an existing ring buffer file to append the new data records, or to create a new file if it does not already exist. An error is not returned if the file is already open.)
- **A_Close** (Closes an open ring buffer file. An error is not returned if the file is already closed.)
- **A_Create** (Opens a new ring buffer file. If the file already exists, it is overwritten. An error is not returned if the file is already open.)
- **A_AddTail** (Write a new data record into the ring buffer file.)
- **A_GetHead** (Reads the oldest data record from the ring buffer file, but does not remove it – the data pointer is not moved to the next data record.)

- **A_RemoveHead** (Reads and removes the oldest data record from the ring buffer file – the data pointer is moved on to the next data record.)
- **A_Reset** (Deletes all the data records from an open ring buffer file. Only the data pointer and the number of data records are reset; the existing physical file size is not changed, although the oldest data records will be overwritten by new ones.)

When a ring buffer file that already exists is opened, the file header is read first. In a ring buffer file that has previously been closed without error, bit 0 in the header status (Header.status.Bit 0) must be zero. If not, it is assumed that the file was not properly closed beforehand, and the corrupt file is replaced by a new, empty file, while Header.status.Bit 1 is set to 1 (file corrupted). When the file is closed, Header.status.Bit 0 is set to 0, and the complete file header is updated in the file.

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetId := '';
    sPathName   : T_MaxString := 'c:\Temp\data.dat';
    ePath       : E_OpenPath := PATH_GENERIC;
    nID         : UDINT := 0;
    cbBuffer    : UDINT := 16#100000;
    bOverwrite  : BOOL := FALSE;
    pWriteBuff  : DWORD;
    cbWriteLen  : UDINT;
    pReadBuff   : DWORD;
    cbReadLen   : UDINT;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: a string containing the network address of the TwinCAT computer where the buffer file is to be written or read can be given here. If it is to be run on the local computer, an empty string can be entered.

sPathName: contains the path and file name of the buffer file to be opened.

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

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

nID: user-defined 32-bit value. When a new file is opened, this value is saved in the file and can be used, for instance, for checking the version of the buffer file.

cbBuffer: the maximum size, in bytes, of the buffer file that is to be open. This parameter is saved in the file header when the file is created, and is checked when the same file is opened again. You can only reopen files that have been created using the same maximum buffer size. You cannot, in other words, create a file with a smaller buffer size, fill it with data records, and then open it again with a larger buffer size. If the check of the maximum buffer size fails, a new file with the new buffer size is automatically created and opened. Bit 1 (file corrupted) is also set in the file header status.

bOverwrite: write behavior when the maximum file buffer size is reached. If this input is TRUE, the oldest entries are overwritten if the max. file buffer size has already been reached (entries are deleted until the free buffer size is sufficient to store the new entry). If this input is FALSE, a buffer overflow when the maximum file buffer size is reached is reported as an error.

pWriteBuff: the address of the PLC variable or of a buffer variable that contains the value data that is to be written. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that *cbWriteLen* data bytes can be taken from it.

cbWriteLen: the number of value data bytes that are to be written (in the case of string variables this includes the terminating null).

pReadBuff: the address of the PLC variables or of a buffer variable into which the value data that has been read is to be copied. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that it can accept cbReadLen data bytes. The size of the buffer variables in bytes must be greater than or equal to the size of the data record that is to be read.

cbReadLen: number of value data bytes to read. If the buffer size is too small, no data is copied, the function block reports a buffer underflow error and the required buffer size for the next record to be read is returned at the *cbReturn* output.

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

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    nErrId      :UDINT;
    cbReturn    :UDINT;
    stHeader    :ST_FileRBufferHead;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until feedback is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the *bBusy* output is reset.

nErrId : Supplies the ADS error number or the command-specific error code (table) when the *bError* output is set.

cbReturn: The number of value data bytes successfully read. If a read buffer underflow error has occurred, this output supplies the necessary read buffer size in bytes.

stHeader: Ring buffer file header [▶ 247]/status.

Command-specific error codes	Error descriptions
0x8000	Writing: File buffer is empty. Reading: File buffer overflow.
0x8001	PLC application: read buffer underflow (pReadBuff, cbReadLen) has been dimensioned too small.
0x8002	Ring buffer file is closed and must be opened first.
0x8003	Input parameter has incorrect value.

It is not essential for the PLC application to know the binary structure of the file. The following illustration, however, shows the general structure of the ring buffer file used:

Header (48 bytes) see the description of ST_FileRBufferHead	Length of data record 1 (4 bytes)	Data record 1	Length of data record 2 (4 bytes)	Data record 2	Length of data record 3 (4 bytes)	Data record 3

An empty ring buffer file only contains the header. The buffer itself follows the header. The variables Header.ptrFirst and Header.ptrLast point to the position immediately behind the header. Writing causes the ptrLast data pointer to be moved onwards. The ptrFirst data pointer follows the ptrLast data pointer during reading. When the maximum buffer size is reached the pointers are returned to the start of the buffer.

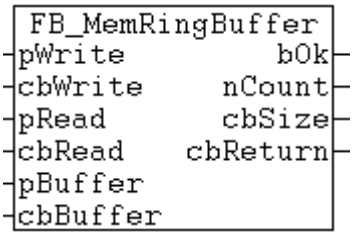
Example:

See: Example: File ring FiFo (FB_FileRingBuffer). [▶ 257]

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build > 1313	PC or CX (x86, ARM)	TcUtilities.Lib

4.32 FB_MemRingBuffer



The FB_MemRingBuffer function block allows data records of varying lengths to be written into a ring buffer, or for data records that have previously been written there to be removed from the ring buffer . The buffer operates as a FIFO; data records are read from it in the same sequence in which they were first written into the ring buffer. This means that the oldest entries are the first ones that are read. The buffer storage is provided to the function block via the input variables *pBuffer* / *cbBuffer*. Writing and reading the data records is controlled by action calls. The function block features the following tasks:

- **A_AddTail** (Write a new data record into the ring buffer.)
- **A_GetHead** (Reads the oldest data record from the ring buffer, but does not remove it.)
- **A_RemoveHead** (Reads and removes the oldest data record from the ring buffer.)
- **A_Reset** (Deletes all the data records in the buffer.)

VAR_INPUT

```
VAR_INPUT
    pWrite   : DWORD;
    cbWrite  : UDINT;
    pRead    : DWORD;
    cbRead   : UDINT;
    pBuffer  : DWORD;
    cbBuffer : UDINT;
END_VAR
```

pWrite: The address of the PLC variable or of a buffer variable that contains the value data that is to be written. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that *cbWrite* data bytes can be taken from it.

cbWrite: The number of value data bytes that are to be written. (In the case of string variables this includes the final null).

pRead: The address of the PLC variables or of a buffer variable into which the value data that has been read is to be copied. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that it can accept cbRead data bytes. The size of the buffer variables in bytes must be greater than or equal to the size of the data record that is to be read.

cbRead: The number of value data bytes to be read. If the buffer size is too small, data is not copied. The function block reports a buffer underflow error (bOk = FALSE), and the buffer size required for the next data record that is to be read is returned at the *cbReturn* output.

pBuffer: Address of PLC variable (e.g. ARRAY[...] OF BYTES) that is to be used by the function block as buffer storage. The address can be determined with the ADR operator.

cbBuffer: The maximum size, in bytes, of the PLC variables that is to be used as buffer storage. The size can be determined by the SIZEOF operator.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk       :BOOL;
    nCount    :UDINT;
    cbSize    :UDINT;
    cbReturn  :UDINT;
END_VAR
```

bOk: Supplies TRUE if a new data record was inserted/deleted successfully. Supplies FALSE if a buffer overflow occurs or if no more entries do exist in the buffer.

nCount: Supplies the current number of buffered data records.

cbSize: Supplies the current number of stored data bytes in the buffer. The number of occupied data bytes is always bigger than the actual number of written value data.Each data record is complemented by addtional information to be localised later.

cbReturn: The number of value data bytes successfully read. If a read buffer underflow error has occurred, this output supplies the necessary read buffer size in bytes. In this case the *cbRead* length is too small.

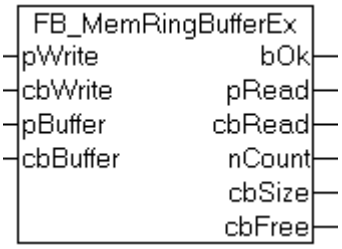
Example:

See: Example: Memory ring FiFo (FB_MemRingBuffer). [▶ 258]

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build > 1313	PC or CX (x86, ARM)	TcUtilities.Lib

4.33 FB_MemRingBufferEx



The function block FB_MemRingBufferEx can be used to write data sets with different lengths in a ring buffer or to read previously written data sets from the ring buffer. The written data sets are read in the same order as they were previously written to the ring buffer, based on the FIFO principle, i.e. the oldest entries are read first. The buffer memory is made available to the function block via the *pBuffer*/*cbBuffer* input variables. Writing/reading of data sets is controlled via action calls.

The functionality of this function block is similar to that of the FB_MemRingBuffer [▶ 169] function block. During reading of datasets the FB_MemRingBuffer copies the data into an external buffer variable. FB_MemRingBufferEx only provides a reference for the data set (address pointer/length). The application must then copy the data itself for further processing.

The function block features the following tasks:

- **A_AddTail** (writes a new data set into the ring buffer.)
- **A_GetHead** (returns a reference: address pointer/length of the oldest data set in the ring buffer, but does not remove it.)
- **A_FreeHead** (reads and removes the oldest data set from the ring buffer. The returned address pointer/length is zero! The free memory segment is released for a new data set.)
- **A_reset** (deletes all data sets in the ring buffer.)
- **A_GetFreeSize** (returns the byte size of the largest free memory segment in the buffer)

VAR_INPUT

```
VAR_INPUT
    pWrite    : DWORD;
    cbWrite   : UDINT;
    pBuffer   : DWORD;
    cbBuffer  : UDINT;
END_VAR
```

pWrite: The address of the PLC variable or of a buffer variable that contains the value data that is to be written. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that *cbWrite* data bytes can be taken from it.

cbWrite: The number of value data bytes that are to be written. (In the case of string variables this includes the final null). The size can be determined with the SIZEOF operator.

pBuffer: Address of a PLC variable (e.g. ARRAY[...] [...] OF BYTES) to be used as buffer memory by the function block. The address can be determined with the ADR operator.

cbBuffer: Max. byte size of the PLC variable to be used as buffer memory. The size can be determined with the SIZEOF operator.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     : BOOL;
    pRead   : DWORD;
    cbRead  : UDINT;
    nCount  : UDINT;
    cbSize  : UDINT;
    cbFree  : UDINT;
END_VAR
```

bOk: Returns TRUE, if a new data set was added or removed successfully. Returns FALSE in the event of a buffer overflow or if no more entries are available in the buffer.

pRead: After a call of the action: *A_GetHead*, if successful (bOk=TRUE) this variable returns a reference (address pointer) for the oldest data set in the ring buffer. It returns zero if no more data sets are available in the ring buffer.

cbRead: After a call of the action: *A_GetHead*, if successful (bOk=TRUE) this variable returns the length of the oldest data set in the ring buffer. It returns zero if no more data sets are available in the ring buffer.

nCount: Returns the current number of queued data sets.

cbSize: Returns the current number of assigned data bytes in the buffer. The number of assigned data bytes is always greater than the actual number of written value data. Each data set is complemented with additional information, so that it can be located later.

cbFree: After a call of the action: *A_GetFreeSize*, returns the byte size of the largest free memory segment in the buffer. The data sets must use continuous addresses in the buffer memory, because the function block returns a reference for the data sets. This automatically leads to segmentation at the end of the buffer. This memory cannot be used if the new data set is greater then the free segment at the end of the buffer.

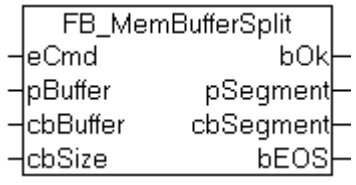
Example:

See: Example: Memory ring FiFo (FB_MemRingBufferEx). [▶ 260]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

4.34 FB_MemBufferSplit



This function block splits a memory area (data buffer) into several smaller segments of certain maximum length as required. The function block returns a smaller partial segment, if the length of the last segment is smaller than required.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
    cbSize      : UDINT;
END_VAR
```

eCmd: Control parameter [▶ 235] for the enumeration block: eEnumCmd_First returns the first segment, eEnumCmd_Next returns the next segment. No other parameters are used.

pBuffer: Address (pointer) of the data buffer to be divided. The address can be determined with the ADR operator.

cbBuffer: Length of the data buffer to be divided. The length can be determined with the SIZEOF operator.

cbSize: Maximum segment size into which the data buffer is to be divided.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk       : BOOL;
    pSegment  : DWORD;
    cbSegment : UDINT;
    bEOS      : BOOL;
END_VAR
```

bOk: TRUE = success, FALSE = error, invalid parameter value or no further segment available.

pSegment: Address (pointer) to the next data segment.

cbSegment: Length (bytes) of the next data segment.

bEOS: End of segment. TRUE = last segment. FALSE = further segments follow.

Example in ST:

In the following example the *buffer* variable is divided into 5-byte segments. The returned segments are converted to a hexadecimal string for test purposes.

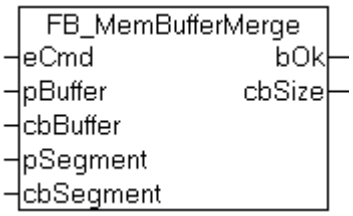
```
PROGRAM MAIN
VAR
    bSplit  : BOOL := TRUE;
    buffer  : ARRAY[1..30] OF BYTE := 16#A,1,2,3,4,5,6,7,8,9,16#B,1,2,3,4,5,6,7,8,9,16#C,1,2,3,4,5,6
,7,8,9;
    fbSplit     : FB_MemBufferSplit;
    sHex    : T_MaxString;
END_VAR
```

```
IF bSplit THEN
    bSplit := FALSE;
    fbSplit.eCmd := eEnumCmd_First;

    REPEAT
        fbSplit( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), cbSize := 5 );
        IF fbSplit.bOk THEN

            sHex := DATA_TO_HEXSTR( pData := fbSplit.pSegment, cbData := fbSplit.cbSegment, FALSE );

            fbSplit.eCmd := eEnumCmd_Next;
        END_IF
    UNTIL NOT fbSplit.bOk
    END_REPEAT
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1331	PC or CX (x86, ARM)	TcUtilities.Lib

4.35 FB_MemBufferMerge



This function block consolidates individual smaller data segments to form a larger data segment. The target buffer must be transferred as input parameter to the block. No further data bytes are added, if the segment to be added exceeds the remaining free buffer size.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
    pSegment    : DWORD := 0;
    cbSegment   : UDINT := 0;
END_VAR
```

eCmd: Control parameter [▶ 235] for the enumeration block: eEnumCmd_First adds the first segment, eEnumCmd_Next adds the next segment. No other parameters are used.

pBuffer: Address (pointer) for the target buffer variable. The address can be determined with the ADR operator.

cbBuffer: Maximum available size (in bytes) of the target buffer variables. The size can be determined with the SIZEOF operator.

pSegment: Address (pointer) for the next data segment to be added (optional, may be zero). The address can also be determined with the ADR operator.

cbSegment: Size of the next data segment to be added (optional, may be zero). The size can also be determined with the SIZEOF operator.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     : BOOL;
    cbSize  : UDINT;
END_VAR
```

bOk: TRUE = success, FALSE = buffer overflow or faulty input parameters.

cbSize: Current buffer fill status (number of data bytes in the buffer).

Example in ST:

In the following example, the large data segment is converted to a hexadecimal string after the smaller data segments have been consolidated (for test purposes).

```
PROGRAM MAIN
VAR
    bMerge  : BOOL := TRUE;
    fbMerge : FB_MemBufferMerge;
    buffer  : ARRAY[0..25] OF BYTE;
    seg1    : ARRAY[0..5] OF BYTE := 0,1,2,3,4,5;
    seg2    : ARRAY[0..3] OF BYTE := 6,7,8,9;
```

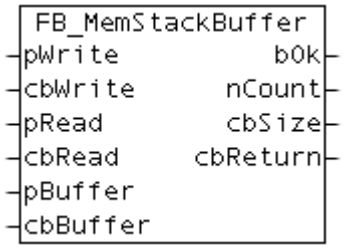
```
    seg3    : ARRAY[0..9] OF BYTE := 10,11,12,13,14,15,16,17,18,19;
    sHex    : T_MaxString;
END_VAR
```

```
IF bMerge THEN
    bMerge := FALSE;

    fbMerge( eCmd := eEnumCmd_First, pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pSegment :=
 ADR(seg1), cbSegment:= SIZEOF(seg1) );
    fbMerge( eCmd := eEnumCmd_Next, pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pSegment :=
ADR(seg2), cbSegment:= SIZEOF(seg2 ) );
    fbMerge( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pSegment := ADR(seg3), cbSegment:=
SIZEOF(seg3) );
    fbMerge( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pSegment := 0, cbSegment:= 0 );
(* merge zero length segment *)
 fbMerge( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pSegment := ADR(seg3), cbSegment:= SIZ
EOF(seg3) );
    IF NOT fbMerge.bOk THEN
        ;(* TODO: Error handler *)
    END_IF

    sHex := DATA_TO_HEXSTR( pData := ADR(buffer), cbData := fbMerge.cbSize, FALSE );
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1331	PC or CX (x86, ARM)	TcUtilities.Lib

4.36 FB_MemStackBuffer



The FB_MemStackBuffer function block allows data records of varying lengths to be written into a buffer, or for data records that have previously been written there to be removed from the buffer. The buffer operates as a LIFO (Last In - First Out). Data records are read from it in the reverse sequence in which they were first written into the buffer. This means that the newest entries are the first ones that are read. The buffer storage is provided to the function block via the input variables *pBuffer* and *cbBuffer*. Writing and reading the data records is controlled by action calls. The function block features the following tasks:

- **A_Push():** Write a new data record into the buffer;
- **A_Top():** Reads the newest/latest data record from the buffer, but does not remove it;
- **A_Pop():** Reads and removes the newest/latest data record from the buffer;
- **A_Reset():** Deletes all the data records in the buffer;

VAR_INPUT

```
VAR_INPUT
    pWrite   : DWORD;
    cbWrite  : UDINT;
    pRead    : DWORD;
    cbRead   : UDINT;
    pBuffer  : DWORD;
    cbBuffer : UDINT;
END_VAR
```

pWrite: The address of the PLC variable or of a buffer variable that contains the value data that is to be written. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that *cbWrite* data bytes can be taken from it.

cbWrite: The number of value data bytes that are to be written. (In the case of string variables this includes the final null).

pRead: The address of the PLC variables or of a buffer variable into which the value data that has been read is to be copied. The address can be determined with the ADR operator. The programmer is himself responsible for dimensioning the buffer variable in such a way that it can accept cbRead data bytes. The size of the buffer variables in bytes must be greater than or equal to the size of the data record that is to be read.

cbRead: The number of value data bytes to be read. If the buffer size is too small, data is not copied. The function block reports a buffer underflow error (bOk = FALSE), and the buffer size required for the next data record that is to be read is returned at the *cbReturn* output.

pBuffer: Address of PLC variable (e.g. ARRAY[...] OF BYTES) that is to be used by the function block as buffer storage. The address can be determined with the ADR operator.

cbBuffer: The maximum size, in bytes, of the PLC variables that is to be used as buffer storage. The size can be determined by the SIZEOF operator.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     :BOOL;
    nCount  :UDINT;
    cbSize  :UDINT;
    cbReturn    :UDINT;
END_VAR
```

bOk: Supplies TRUE if a new data record was inserted/deleted successfully. Supplies FALSE if a buffer overflow occurs or if no more entries do exist in the buffer.

nCount: Supplies the current number of buffered data records.

cbSize: Supplies the current number of stored data bytes in the buffer. The number of occupied data bytes is always bigger than the actual number of written value data.Each data record is complemented by addtional information to be localised later.

cbReturn: The number of value data bytes successfully read. If a read buffer underflow error has occurred, this output supplies the necessary read buffer size in bytes. In this case the *cbRead* length is too small.

Example:

The following example illustrates a simple application of the function block. Strings with different length should be buffered. A rising edge at *bReset* deletes all buffer entries. If you set *bAdd* = TRUE a ten new string values will be written into the buffer. With a rising edge at *bRemove* the newest/latest string is removed. With a rising edge at *bGet* the newest/latest string is read but not removed.

Declaration:

```
PROGRAM MAIN
VAR
    buffer  : ARRAY[0..1000] OF BYTE;
    fbStack : FB_MemStackBuffer;

    bReset  : BOOL := TRUE;
    bAdd    : BOOL := TRUE;
    bGet    : BOOL := TRUE;
    bRemove : BOOL := TRUE;

    putEntry    : ARRAY[0..9] OF STRING(20) := 'Str_1', 'Str_2', 'Str_3', 'Str_4', 'Str_5', 'Str_6',
 'Str_7', 'Str_8', 'Str_9', 'Str_10';
    getEntry    : STRING;
    i       : UDINT;
END_VAR
```

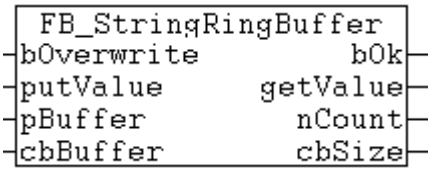
Program:

```
IF bReset THEN(* Clear buffer *)
    bReset := FALSE;
    fbStack.A_Reset( pBuffer := ADR( buffer ), cbBuffer := SIZEOF( buffer ) );
END_IF

IF bAdd THEN(* Add entries *)
    bAdd := FALSE;
    FOR i:= 0 TO 9 BY 1 DO
        fbStack.A_Push( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pWrite := ADR(putEntry[i
]), cbWrite := LEN(putEntry[i]) + 1 );
        IF fbStack.bOk THEN(* Success *)
            ;
        ELSE(* Buffer overflow *)
            ;
        END_IF
    END_FOR
END_IF

IF bGet THEN(* Peek newest entry *)
    bGet := FALSE;
    fbStack.A_Top( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pRead := ADR(getEntry), cbRea
d := SIZEOF(getEntry) );
    IF fbStack.bOk THEN(* Success *)
        ;
    ELSE(* Buffer is empty *)
        ;
    END_IF
END_IF

IF bRemove THEN(* Remove newest entry *)
    bRemove := FALSE;
    fbStack.A_Pop( pBuffer := ADR(buffer), cbBuffer := SIZEOF(buffer), pRead := ADR(getEntry), cbRea
d := SIZEOF(getEntry) );
    IF fbStack.bOk THEN(* Success *)
        ;
    ELSE(* Buffer is empty *)
        ;
    END_IF
END_IF
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.11.0 Build > 2211	PC or CX (x86, ARM)	TcUtilities.Lib

4.37 FB_StringRingBuffer



The FB_StringRingBuffer function block allows strings to be written into a ring buffer, or for strings that have previously been written there to be removed from the ring buffer . The buffer operates as a FIFO; data records are read from it in the same sequence in which they were first written into the ring buffer. This means that the oldest entries are the first ones that are read. The buffer storage is provided to the function block via the input variables *pBuffer* / *cbBuffer*. Writing and reading the strings is controlled by action calls. The function block features the following tasks:

- **A_AddTail** (Write a new string into the ring buffer.)
- **A_GetHead** (Reads the oldest string from the ring buffer, but does not remove it)
- **A_RemoveHead** (Reads and removes the oldest string from the ring buffer)
- **A_Reset** (Deletes all the strings in the buffer).

Internal the FB_MemRingBuffer function block is used. See also the description of the function block: FB_MemRingBuffer [▶ 169].

VAR_INPUT

```
VAR_INPUT
    bOverwrite  : BOOL;
    putValue    : T_MaxString := '';
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
END_VAR
```

bOverwrite: If TRUE and buffer overflow => overwrite the oldest entry. If FALSE and buffer overflow => report error (bOk = FALSE).

putValue: String value to be written.

pBuffer: Address of PLC variable (e.g. ARRAY[...] OF BYTES) that is to be used by the function block as buffer storage. The address can be determined with the ADR operator.

cbBuffer: The maximum size, in bytes, of the PLC variables that is to be used as buffer storage. The size can be determined by the SIZEOF operator.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk       : BOOL;
    getValue  : T_MaxString := '';
    nCount    : UDINT;
    cbSize    : UDINT;
END_VAR
```

bOk: Supplies TRUE if a new data record was inserted/deleted successfully. Supplies FALSE if a buffer overflow occurs or if no more entries do exist in the buffer.

getValue: This output contains the recent read string value.

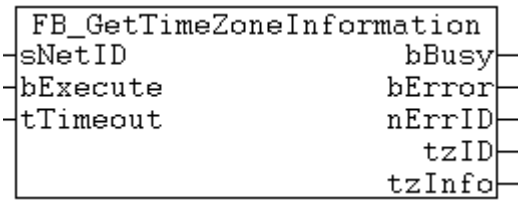
nCount: Supplies the current number of buffered data records.

cbSize: Supplies the current number of stored data bytes in the buffer. The number of occupied data bytes is always bigger than the actual number of written value data.Each data record is complemented by addtional information to be localised later.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

4.38 FB_GetTimeZoneInformation



With this function block the time zone settings of the operating system can be read.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the network address of the TwinCAT Computer can be given, whose time zone setting is to be read. The string can also be empty for the local computer.

bExecute: The command is executed with a positive edge at the input

tTimeOut: Maximum time allowed for the execution of the ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy    : BOOL;
    bError   : BOOL;
    nErrId   : UDINT;
    tzID     : E_TimeZoneID;
    tzInfo   : ST_TimeZoneInformation;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrId: When the bError output is set, this variable supplies the ADS error or the command-specific error code (table).

tzID: Additional daylight-saving/standard time information [▶ 237] (not always existent).

tzInfo: In case of success this structure [▶ 247] variable supplies the current time zone information of the operating system.

Example: See description of the function block FB_SetTimeZoneInformation [▶ 178].

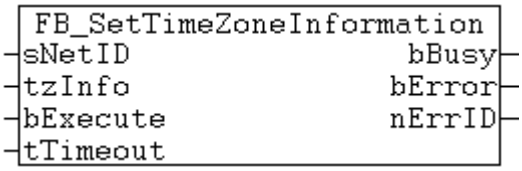
Further time, time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation, FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1319	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.39 FB_SetTimeZoneInformation



This function block can be used to modify or set the time zone settings of the operating system.

The operating system may change some time settings after the new time zone settings were set. The time may have to be reset. The time can be set with the function block: NT_SetLocalTime [▶ 130].

VAR_INPUT

```
VAR_INPUT
    sNetID     : T_AmsNetID;
    tzInfo     : ST_TimeZoneInformation;
    bExecute   : BOOL;
    tTimeout   : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Here a string containing the network address of the TwinCAT Computer can be given, whose time zone setting is to be changed. The string can also be empty for the local computer.

tzInfo: Structure [▶ 247] with the new time zone settings to be set.

bExecute: The command is executed with a positive edge at the input

tTimeOut: Maximum time allowed for the execution of the ADS command.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the execution of the command, then this output is set, after the *bBusy* output has been reset.

nErrId: When the bError output is set, this variable supplies the ADS error or the command-specific error code (table).

Example in ST:

The time zone "West Europa Standard Time" is to be set on the local TwinCAT System. As example the constant **WEST_EUROPE_TZI** with the required parameter values is declared in the PLC library. To configure other time zones the *tzInfo* input of the function block has to be allocated with the accordant values (see description of ST_TimeZoneInformation [▶ 247] structure).

```
VAR_GLOBAL CONSTANT
...
    (* West Europa Standard Time Zone settings *)
    WEST_EUROPE_TZI : ST_TimeZoneInformation := (bias:=-60,
                      standardName:='W. Europe Standard Time',
                      standardDate:=(wYear:=0,wMonth:=10,wDayOfWeek:=0,wDay:=5,wHour:=3),
                      standardBias:=0,
                      daylightName:='W. Europe Daylight Time',
                      daylightDate:=(wYear:=0,wMonth:=3,wDayOfWeek:=0,wDay:=5,wHour:=2),
                      daylightBias:=-60);
...
ENd_VAR
```

Declaration part:

```
PROGRAM MAIN
VAR
    fbGet : FB_GetTimeZoneInformation;
    fbSet : FB_SetTimeZoneInformation;

    tzi_get : ST_TimeZoneInformation;
    tzID : E_TimeZoneID;

    bGet : BOOL := TRUE;
    bSet : BOOL := FALSE;
END_VAR
```

The desired time zone setting is set with a rising edge at the *bSet* variable. To check the current settings the settings can be read with a rising edge at the *bGet* variable.

```
IF bGet THEN
    bGet := FALSE;
    fbGet(bExecute := TRUE);
ELSE
    fbGet(bExecute := FALSE, tzInfo => tzi_get, tzID => tzID );
END_IF

IF bSet THEN
    bSet := FALSE;
    fbSet( bExecute := TRUE, tzInfo := WEST_EUROPE_TZI );
ELSE
    fbSet( bExecute := FALSE );
END_IF
```

Further time, time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation, NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

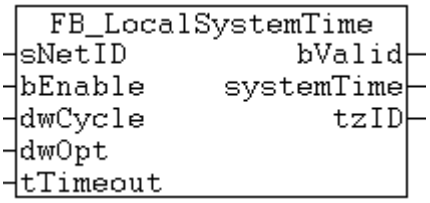
Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1319	PC or CX (x86) CX (ARM)	TcUtilities.Lib

Also see about this

- E_TimeZoneID [▶ 237]

4.40 FB_LocalSystemTime



In some applications the local Windows system time is synchronized via the SNTP time server or a radio clock. The local Windows system time (the local Windows system time is displayed in the task bar) often has to be used in the PLC (e.g. for time stamp log messages to the HMI). For such applications the FB_LocalSystemTime function block can be useful.

This function block internally combines the functions of the following function blocks: RTC_EX2 [▶ 214], NT_GetTime [▶ 129], FB_GetTimeZoneInformation [▶ 177] and NT_SetTimeToRTCTime [▶ 133]. The RTC_EX2 block can be used for generating time stamps for log outputs, for example. However, this block has the disadvantage that its time is not synchronised with the local Windows system time and must be resynchronised cyclically via the NT_GetTime function block (see RTC block examples in the documentation). Cyclic synchronisation of the internal time (*systemTime* output) is already implemented in the function block. The cycle time can be configured via the *dwCycle* input. The function block also provides time zone information (daylight-saving/standard time).

The FB_LocalSystemTime function block must be called cyclically (e.g. every second or during each PLC cycle). This is necessary to enable the time between synchronisations to be calculated.

Imprecise system clock

The local Windows system time is read with the aid of acyclic services (ADS function blocks). Due to the system characteristics the runtime of the ADS commands cannot be specified/estimated. Differences in command runtimes may lead to time jitter at the systemTime output, depending on the operating system, the synchronization interval, and the PLC cycle time. For this reason, the time supplied by the block is only suitable to a limited extent for measuring tasks requiring higher precision, although it tends to be adequate for building automation applications, for example.

Switching between daylight-saving time and standard time

The function block cannot be called exactly at the time when the switchover from daylight-saving time to standard time and vice versa takes place. In order to avoid complex calculations, the following implementation was chosen (explained in the example).

In our example the function block synchronises its own time with the local Windows system time (grey) every 60 seconds.
The PLC application needs and reads the time in the function block (e.g. every 30 seconds, blue). In our example the switchover between daylight-saving time and standard time is detected with a delay of 15 seconds. This behaviour should be unproblematic for most applications.

Switching from standard time to daylight-saving time

- ...
- 30-03-2008-01:58:10, tzID = standard time
- 30-03-2008-01:58:15, after internal synchronisation
- 30-03-2008-01:58:40, tzID = standard time
- 30-03-2008-01:59:10, tzID = standard time
- 30-03-2008-01:59:15, after internal synchronisation
- 30-03-2008-01:59:40, tzID = standard time
- 30-03-2008-02:00:00, the operating system changes the time from 2 am to 3 am
- 30-03-2008-02:00:10, tzID = standard time (still!)
- 30-03-2008-03:00:15, after internal synchronisation, subsequent tzID = daylight-saving time
- 30-03-2008-03:00:40, tzID = daylight-saving time
- 30-03-2008-03:01:10, tzID = daylight-saving time
- 30-03-2008-03:01:15, after internal synchronisation
- 30-03-2008-03:01:40, tzID = daylight-saving time
- ...

Switching from daylight-saving time to standard time

- ...
- 26-10-2008-02:58:10, tzID = daylight-saving time
- 26-10-2008-02:58:15, after internal synchronisation
- 26-10-2008-02:58:40, tzID = daylight-saving time
- 26-10-2008-02:59:10, tzID = daylight-saving time
- 26-10-2008-02:59:15, after internal synchronisation
- 26-10-2008-02:59:40, tzID = daylight-saving time
- 26-10-2008-03:00:00, the operating system changes the time from 3 am to 2 am
- 26-10-2008-03:00:10, tzID = daylight-saving time (still!)
- 26-10-2008-02:00:15, after internal synchronisation, subsequent tzID = standard time
- 26-10-2008-02:00:40, tzID = standard time
- 26-10-2008-02:01:10, tzID = standard time
- 26-10-2008-02:01:15, after internal synchronisation
- 26-10-2008-02:01:40, tzID = standard time

- ...

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID := '';
    bEnable     : BOOL;
    dwCycle     : DWORD(1..86400) := 5;
    dwOpt       : DWORD := 1;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: A string containing the network address of the TwinCAT computer whose time is to be used for the synchronization can be given here. If it is to be run on the local computer, an empty string can be entered.

bEnable: A rising edge at this input triggers immediate synchronization of the internal time with the local Windows system time. The output *bValid* remains set to FALSE until the synchronization is complete. The first rising edge activates the cyclic synchronization. The subsequent cyclic synchronizations are then executed automatically. In most cases the application only must set this input to TRUE once.

dwCycle: Cycle time (in seconds) during which the function block resynchronizes its own time. Cyclic synchronization is activated after the first rising edge at the *bEnable* input. Default: Synchronization every 5 seconds.

dwOpt: Additional option parameters. The following parameters are currently available:

- Bit 0: If this is set, the Windows system time is additionally synchronized cyclically with the hardware clock (RTC) (corresponds to the function NT_SetTimeToRTCTime). Default: Activated.
  This option is irrelevant for a Windows CE system.

tTimeout: States the length of the timeout that may not be exceeded by execution of the internal ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    bValid      : BOOL;
    systemTime  : TIMESTRUCT;
    tzID        : E_TimeZoneID := eTimeZoneID_Invalid;
END_VAR
```

bValid: The time at the *systemTime* output is invalid if this output is FALSE. The time is valid if TRUE (i.e. it was synchronised with the local Windows time at least once).

systemTime: Local Windows system time [▶ 230].

tzID: Time zone information [▶ 237] (daylight-saving/standard time).

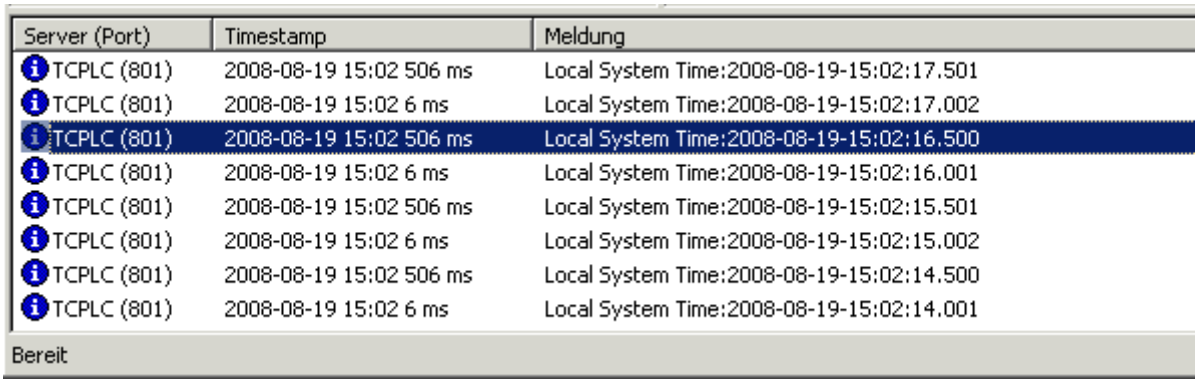
Example in ST:

In the example the FB_LocalSystemTime function block is activated on program startup (rising edge at *bEnable* input). Once the time has been synchronised (*bValid* = TRUE), the PLC writes a message to the TwinCAT System Logview every 500 ms. The internal synchronisation is carried out every second.

```
PROGRAM MAIN
VAR
    fbTime : FB_LocalSystemTime := ( bEnable := TRUE, dwCycle := 1 );

    logTimer : TON := ( IN := TRUE, PT := T#500ms );
END_VAR
```

```
fbTime();

logTimer( IN := fbTime.bValid );
IF logTimer.Q THEN
    logTimer( IN := FALSE ); logTimer( IN := fbTime.bValid );
    ADSLOGSTR( ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_LOG, 'Local System Time:
%s', SYSTEMTIME_TO_STRING(fbTime.systemTime));
END_IF
```

The written messages can be viewed in the TwinCAT System Manager Logview.



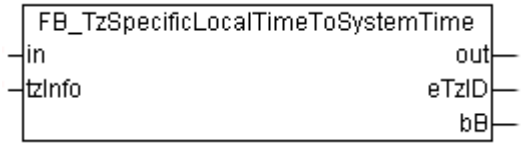
Further time, time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.41 FB_TzSpecificLocalTimeToSystemTime



The function block converts the local time (structured system time format) to UTC time (structured system time format), taking into account the specified time zone information. The function block FB_TzSpecificLocalTimeToFileTime [▶ 184] has similar functionality but uses a different time format (file time format).

The function block is only suitable for conversion of **continuous** local timestamp information. Step changes in local time caused by summer/winter time changeover are permitted and are correctly detected by the function block. Arbitrary changes in local time result in incorrect conversion. The reason: the last converted time is stored internally in function block in order to be able to identify the summer time/winter time information and the B times (see below) when the local time is reset. The function block is associated with an action: A_Reset(). If this action is called the function block outputs and the locally stored (last converted) time are reset to zero.

The step changes in the local time are problematic, since they have to be converted to a linear UTC time. It is therefore advisable to use the (continuous) UTC time for time stamping tasks and to convert the time to respective local time only for display purposes (e.g. in a visualization).

Further information can be found in the documentation for the FB_TzSpecificLocalTimeToFileTime [▶ 184] function block.

VAR_INPUT

```
VAR_INPUT
    in     : TIMESTRUCT;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: Local time (structured system time format [▶ 230]) to be converted.

tzInfo: Structure [▶ 247] variable with the current time zone information of the operating system.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : TIMESTRUCT;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
END_VAR
```

out: Converted UTC time (structured system time format [▶ 230]).

eTzID: Additional daylight-saving/standard time information [▶ 237].

bB: TRUE => B time (e.g.:**02:05:00 CET B**), FALSE => other time (e.g.: **02:05:00 CEST A**). This output is set if the local time jumps back and is reset once the duplicate local time has passed.

Example:

```
PROGRAM MAIN
VAR
    in : TIMESTRUCT := ( wYear := 2011, wMonth := 4, wDay := 29, wHour := 16, wMinute := 46, wSecond
 := 31, wMilliseconds := 99 );(* Local time *)
    out : TIMESTRUCT; (* UTC ttime result is:= ( wYear := 2011, wMonth := 4, wDay := 29, wHour := 14
, wMinute := 46, wSecond := 31, wMilliseconds := 99 ) *)
    fbToUTC : FB_TzSpecificLocalTimeToSystemTime;
END_VAR
fbToUTC( in := in, tzInfo := WEST_EUROPE_TZI, out => out );
```

Further time and time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime, FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

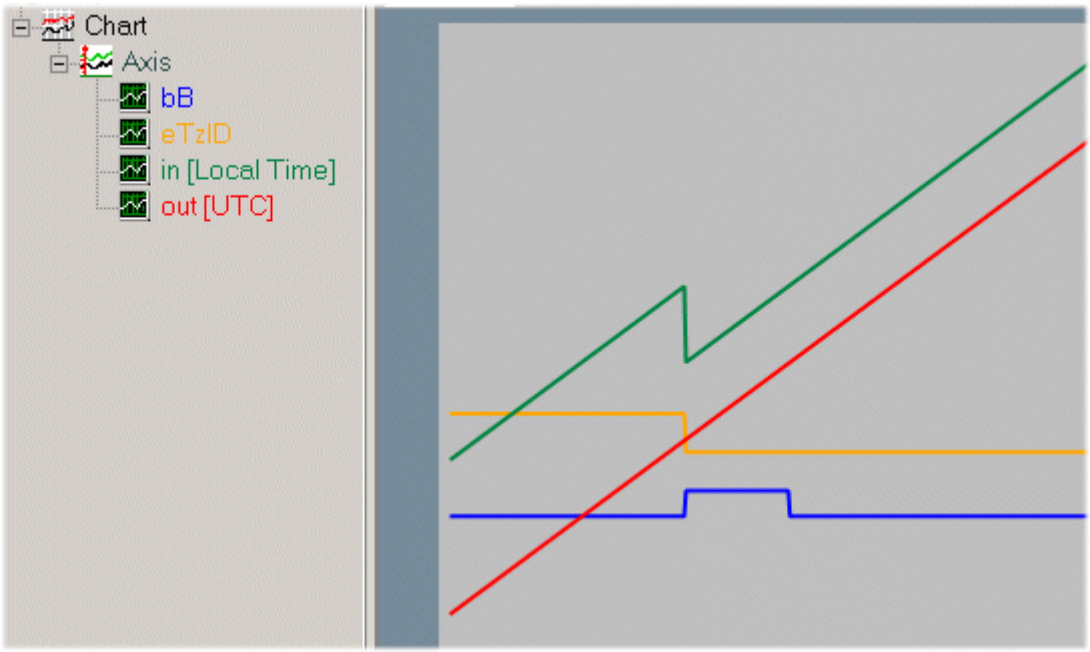
4.42 FB_TzSpecificLocalTimeToFileTime

The function block converts the local time (file time format) to UTC time (file time format), considering the specified time zone information. The function block FB_TzSpecificLocalTimeToSystemTime [▶ 183] has similar functionality but uses a different time format (structured system time format).

The block is only suitable for conversion of **continuous** local timestamp information. Step changes in local time caused by daylight-saving/standard time changeover are permitted and are correctly detected by the block. Arbitrary changes in local time result in incorrect conversion. The reason: the last converted time is stored internally in block to be able to identify the daylight-saving/standard time information and the B times (see below) when the local time is reset. The block is associated with an action: A_Reset(). If this action is called the block outputs and the locally stored (last converted) time are reset to zero.

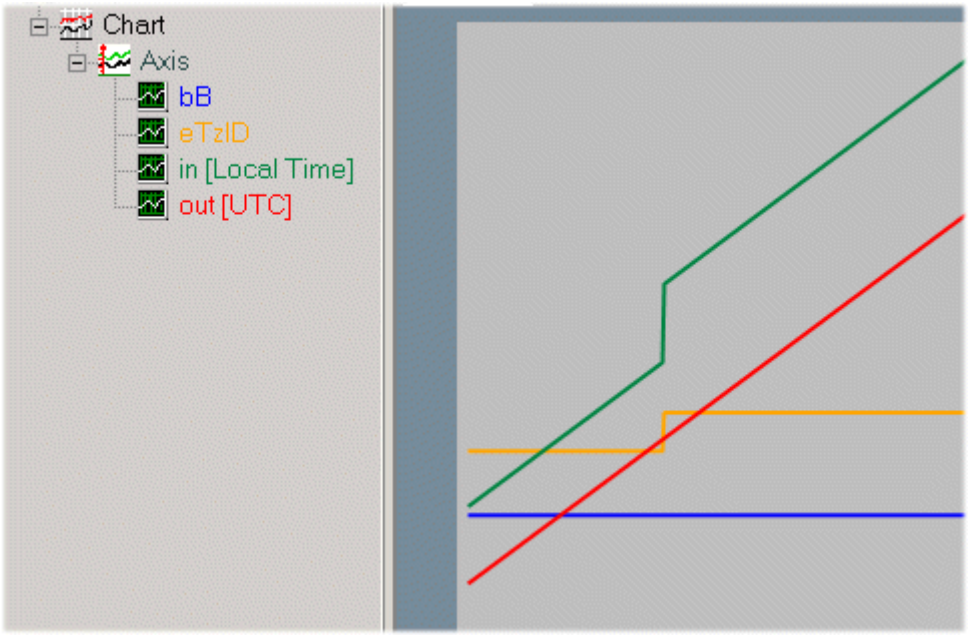
The step changes in the local time are problematic since they must be converted to a linear UTC time. It is therefore advisable to use the (continuous) UTC time for timestamping tasks and to convert the time to respective local time only for display purposes (e.g. in a visualization).

1. Graphic representation of the changeover from daylight-saving time to standard time (tzInfo = WEST_EUROPE_TZI):



The local time (green) jumps back. The UTC output time (red) continuous. The local time: **02h:59m:59s:999ms..** is directly followed by: **02h:00m:00s:000ms..** The times between 2h and 3h occur twice. The duplicate time before the changeover is referred to as **02:05:00 CEST A** , for example, the time after the changeover as **02:05:00 CET B**. The output variable *bB* indicates whether it is the first or the second *pass*. During the second *pass* the *bB* output variable (blue) is set to TRUE. The *bB* output variable is automatically reset once the duplicate time has passed. The time zone ID (orange) changes from *eTimeZoneID_Daylight* (daylight-saving time) to *eTimeZoneID_Standard* (standard time).

2. Graphic representation of the change-over from standard time to daylight-saving time (tzInfo = WEST_EUROPE_TZI):



The local time (green) jumps forward. The UTC output time (red) continuous. The local time: **2h:59m:59s:999ms..** is directly followed by: **3h:00m:00s:000ms..** The time zone ID (orange) changes from *eTimeZoneID_Standard* (standard time) to *eTimeZoneID_Daylight* (daylight-saving time).

VAR_INPUT

```
VAR_INPUT
    in   : T_FILETIME;
    info : ST_TimeZoneInformation;
END_VAR
```

in: Local time (file time format [▶ 239]) to be converted.

tzInfo: Structure [▶ 247] variable with the current time zone information of the operating system.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : T_FILETIME;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
ND_VAR
```

out: Converted UTC time (file time format [▶ 239]).

eTzID: Additional daylight-saving/standard time information [▶ 237].

bB: TRUE => B time (e.g.:**02:05:00 CET B**), FALSE => other time (e.g.: **02:05:00 CEST A**). This output is set if the local time jumps back and is reset once the duplicate local time has passed.

Example:

The local time: DT#2011-09-02-11:01:31 is converted UTC time: DT#2011-09-02-09:01:31.

```
PROGRAM MAIN
VAR
    in : DT := DT#2011-09-02-11:01:31;(* Local time *)
    out : DT;(* UTC time *)
    fbToUTC : FB_TzSpecificLocalTimeToFileTime;
END_VAR
```

```
fbToUTC( in := DT_TO_FILETIME( in ), tzInfo := WEST_EUROPE_TZI );
out := FILETIME_TO_DT( fbToUTC.out );
```

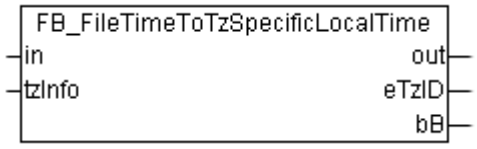
Further time and time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime, FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

4.43 FB_FileTimeToTzSpecificLocalTime

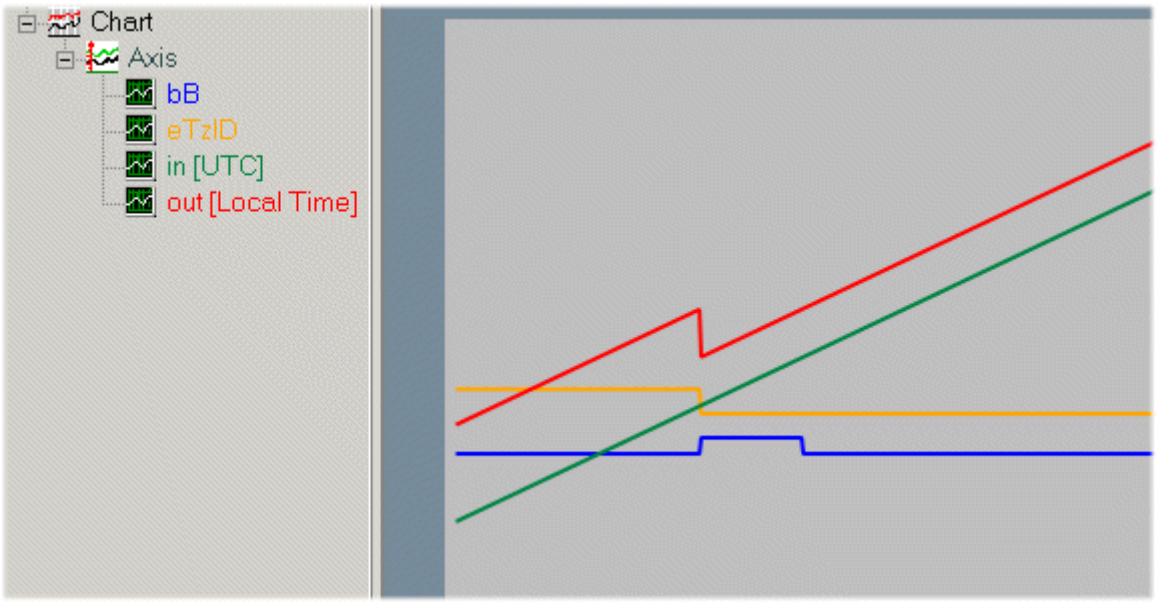


The function block converts the UTC time (file time format) to local time (file time format), taking into account the specified time zone information. The function block FB_SystemTimeToTzSpecificLocalTime [▶ 189] has similar functionality but uses a different time format (structured system time format).

The block is only suitable for conversion of **continuous** UTC timestamp information. The function block uses the time zone information to calculate the required time steps (daylight-saving/standard time changeover) in local time. Time steps in UTC input time are not permitted and lead to incorrect conversion. The reason: the block stores the last converted time internally, so that it can detect the B times (see below) from the UTC input time and the stored value when the local time is changed.

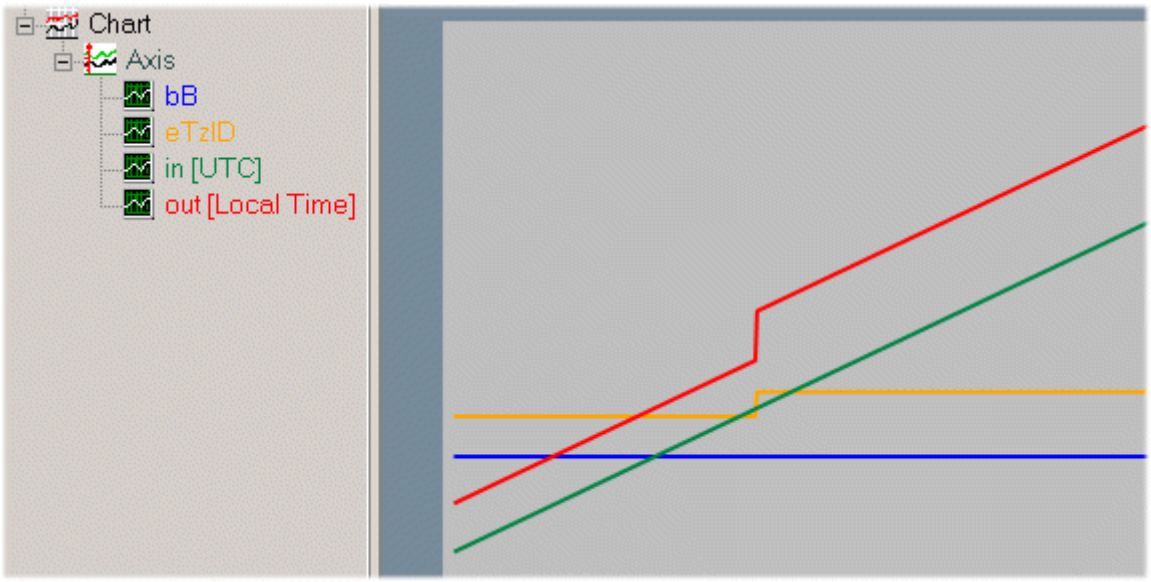
The block is associated with an action: A_Reset(). If this action is called the block outputs and the locally stored (last converted) time are reset to zero.

1. Graphic representation of the changeover from daylight-saving time to standard time (tzInfo = WEST_EUROPE_TZI):



The UTC input time (green) is continuous. The local time (red) jumps back. The local time: **02h:59m:59s:999ms..** is directly followed by: **02h:00m:00s:000ms..** The times between 2h and 3h occur twice. The duplicate time before the changeover is referred to as **02:05:00 CEST A** , for example, the time after the changeover as **02:05:00 CET B**. The output variable *bB* indicates whether it is the first or the second *pass*. During the second *pass* the *bB* output variable (blue) is set to TRUE. The *bB* output variable is automatically reset once the duplicate time has passed. The time zone ID (orange) changes from *eTimeZoneID_Daylight* (daylight-saving time) to *eTimeZoneID_Standard* (standard time).

2. Graphic representation of the change-over from standard time to daylight-saving time (tzInfo = WEST_EUROPE_TZI):



The UTC input time (green) is continuous. The local time (green) jumps forward. The local time: **2h:59m:59s:999ms..** is directly followed by: **3h:00m:00s:000ms..** The time zone ID (orange) changes from *eTimeZoneID_Standard* (standard time) to *eTimeZoneID_Daylight* (daylight-saving time).

VAR_INPUT

```
VAR_INPUT
    in     : T_FILETIME;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: UTC time (file time format [▶ 239]) to be converted.

tzInfo: Structure [▶ 247] variable with the current time zone information of the operating system.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : T_FILETIME;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
END_VAR
```

out: Converted local time (file time format [▶ 239]).

eTzID: Additional daylight-saving/standard time information [▶ 237].

bB: TRUE => B time (e.g.:**02:05:00 CET B**), FALSE => other time (e.g.: **02:05:00 CEST A**). This output is set if the local time jumps back and is reset once the duplicate local time has passed.

Example:

The UTC time: DT#2011-09-02-09:01:31 is converted to local time. The result is: DT#2011-09-02-11:01:31.

```
PROGRAM MAIN
VAR
    in : DT := DT#2011-09-02-09:01:31;(* UTC time *)
    out : DT;(* Local time *)
    fbToLocal : FB_FileTimeToTzSpecificLocalTime;
END_VAR
```

```
fbToLocal( in := DT_TO_FILETIME( in ), tzInfo := WEST_EUROPE_TZI );
out := FILETIME_TO_DT( fbToLocal.out );
```

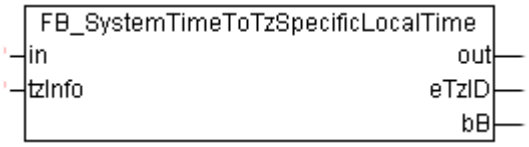
Further time and time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime, FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

4.44 FB_SystemTimeToTzSpecificLocalTime



The function block converts the UTC time (structured system time format) to local time (structured system time format), taking into account the specified time zone information. The function block FB_FileTimeToTzSpecificLocalTime [▶ 187] has similar functionality but uses a different time format (file time format).

The function block is only suitable for conversion of **continuous** UTC timestamp information. The function block uses the time zone information to calculate the required time steps (summer/winter time changeover) in local time. Time steps in UTC input time are not permitted and lead to incorrect conversion. The reason: the function block stores the last converted time internally, so that it can detect the B times (see below) from the UTC input time and the stored value when the local time is changed.

The function block is associated with an action: A_Reset(). If this action is called the function block outputs and the locally stored (last converted) time are reset to zero.

Further information can be found in the documentation for the FB_FileTimeToTzSpecificLocalTime [▶ 187] function block.

VAR_INPUT

```
VAR_INPUT
    in     : TIMESTRUCT;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: UTC time (structured system time format [▶ 230]) to be converted.

tzInfo: Structure [▶ 247] variable with the current time zone information of the operating system.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : TIMESTRUCT;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
END_VAR
```

out: Converted local time (structured system time format).

eTzID: Additional daylight-saving/standard time information.

bB: TRUE => B time (e.g.:**02:05:00 CET B**), FALSE => other time (e.g.: **02:05:00 CEST A**). This output is set if the local time jumps back and is reset once the duplicate local time has passed.

Example:

```
PROGRAM MAIN
VAR
    in : TIMESTRUCT := ( wYear := 2011, wMonth := 4, wDay := 29, wHour := 14, wMinute := 46, wSecond
 := 31, wMilliseconds := 99 );(* UTC time *)
    out : TIMESTRUCT; (* Local time result is:= ( wYear := 2011, wMonth := 4, wDay := 29, wHour := 1
6, wMinute := 46, wSecond := 31, wMilliseconds := 99 ) *)
    fbToLocal : FB_SystemTimeToTzSpecificLocalTime;
END_VAR
```

```
fbToLocal( in := in, tzInfo := WEST_EUROPE_TZI, out => out );
```

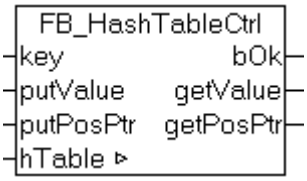
Further time and time zone functions and function blocks:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 184], FB_SystemTimeToTzSpecificLocalTime, FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 178], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 133], F_TranslateFileTimeBias [▶ 62], FB_LocalSystemTime [▶ 180]

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

4.45 FB_HashTableCtrl



The hash table can be used as an efficient tool for finding individual data element among a larger number of data elements. The data objects must have a unique key. The key enables the data objects to be identified unambiguously and found quickly in the table.

The function block FB_HashTableCtrl can be used to realise a simple hash table in the PLC project, using the hashing with chaining (separate chaining) procedure.

The maximum number of data elements cannot be changed at runtime and must be specified in advance. Adding/removing/finding of data elements is controlled through action calls. The function block features the following tasks:

- **A_Add** (adds a new data element to the table (key/value). If an element with the same key already exists, it is overwritten!)
- **A_GetFirst** (reads the first table data element. If successful, *getValue* supplies the associated value.)

- **A_GetNext** (reads the next table data element. The address *putPosPtr* must point to the previous data element!)
- **A_Lookup** (looks for a data element matching the key. If successful, *getValue* supplies the associated value.)
- **A_Remove** (removes a data element matching the key.)
- **A_RemoveAll** (removes all data elements)
- **A_RemoveFirst** (removes the first data element)
- **A_reset** (deletes all data elements and resets the table.)
- **A_GetIndexAtPosPtr (from TwinCAT v2.10 Build >= 1339 and TwinCATv2.11 Build >= 1524)** (returns the array-index data element at address *putPosPtr*. If successfull the output variable *getValue* retuns the zero based array-index number. The value *putValue* is not used!)

VAR_IN_OUT

```
VAR_IN_OUT
    hTable      : T_HHASHTABLE;
END_VAR
```

hTable: Hash table handle [▶ 242]. The handle must be initialized once with the function F_CreateHashTableHnd [▶ 67] before it can be used. A corresponding instance of the handle variable must be created and initialized for each table.

VAR_INPUT

```
VAR_INPUT
    key         : DWORD := 0;
    putValue    : DWORD := 0;
    putPosPtr   : POINTER TO T_HashTableEntry := 0;
END_VAR
```

key: (32 bit). This key enables a data object to be identified and found quickly in the table.

putValue: Value/data element (32 bit).

putPosPtr: Address for a previous data element [▶ 242].

VAR_OUTPUT

```
VAR_OUTPUT
    bOk         : BOOL := FALSE;
    getValue    : DWORD := 0;
    getPosPtr   : POINTER TO T_HashTableEntry := 0;
END_VAR
```

bOk: Returns TRUE if a new data element was added to/removed from or found in the table. Returns FALSE, if the searched data element was not found, the table is empty, or an overflow occurred (table has no free data elements).

getValue: The value matching the key.

getPosPtr: The address for the data element [▶ 242].

Example:

See: Example: Hash table (FB_HashTableCtrl). [▶ 261]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

4.46 FB_LinkedListCtrl

The function block FB_LinkedListCtrl can be used to implement a linked list in the PLC project. A double-linked list is created. A linked list allows values (known as nodes) to be stored. It is possible to iterate the list from the back to the front or the other way. Nodes can quickly be added or deleted.

It is not possible to change the maximum number of nodes at runtime; it must be specified before compiling. An array of type *T_LinkedListEntry* is used as a "node pool". Adding/removing/finding of nodes is controlled through action calls. The function block features the following tasks:

- **A_AddHeadValue** (adds a new node with the value *putValue* to the top of the list. The same value can be added more than once. If successful, *getPosPtr* returns the address while *getValue* returns the value of the new node.)
- **A_AddTailValue** (adds a new node with the value *putValue* to the end of the list. The same value can be added more than once. If successful, *getPosPtr* returns the address while *getValue* returns the value of the new node.)
- **A_FindNext** (searches for the next node (relative to *putPosPtr*) whose value is the same as *putValue*. If successful, *getPosPtr* returns the address while *getValue* returns the value of the node.)
- **A_FindPrev** (searches for the previous node (relative to *putPosPtr*) whose value is the same as *putValue*. If successful, *getPosPtr* returns the address while *getValue* returns the value of the node.)
- **A_GetNext** (navigates to the next node (relative to *putPosPtr*). The address *putPosPtr* must point the previous node! The value of *putValue* is not used.)
- **A_GetPrev** (navigates to the previous node (relative to *putPosPtr*) in the opposite direction to *A_GetNext*. The address *putPosPtr* must point the previous node! The value of *putValue* is not used.)
- **A_GetHead** (reads the starting node. If successful, *getPosPtr* returns the address of the node while *getValue* returns the associated value. The values of *putValue* and *putPosPtr* are not used.)
- **A_GetTail** (reads the final node. If successful, *getPosPtr* returns the address of the node while *getValue* returns the associated value. The values of *putValue* and *putPosPtr* are not used.)
- **A_RemoveHeadValue** (removes a node from the top of the list. If successful, *getPosPtr* returns the address while *getValue* returns the value of the node. The values of *putValue* and *putPosPtr* are not used.)
- **A_RemoveTailValue** (removes a node from the end of the list. If successful, *getPosPtr* returns the address while *getValue* returns the value of the node. The values of *putValue* and *putPosPtr* are not used.)
- **A_RemoveValueAtPosPtr** (searches for and removes a node with address *putPosPtr*. If successful, *getPosPtr* returns the address while *getValue* returns the value of the node. The value of *putValue* is not used.)
- **A_GetIndexAtPosPtr** (returns the array index (from the "node pool") of the node at address *putPosPtr*. When successful, *getValue* returns the null-based array index. The value of *putValue* is not used. Please note that the value of getValue is a node index, not the value of the node!)
- **A_SetValueAtPosPtr** (updates/sets the value of the node with *putValue* at the address *putPosPtr*. If successful, *getPosPtr* returns the address while *getValue* returns the value of the node.)
- **A_Reset** (deletes all list elements and resets the list.)

VAR_IN_OUT

```
VAR_IN_OUT
    hList       : T_HLINKEDLIST;
END_VAR
```

hList: Linked List Handle [▶ 242]. The handle must be initialised once with the function F_CreateLinkedListHnd [▶ 68] before being used. An associated instance of the handle variable must be created and initialised for each linked list

VAR_INPUT

```
VAR_INPUT
    putValue        : DWORD := 0;
    putPosPtr       : POINTER TO T_LinkedListEntry := 0;
END_VAR
```

putValue: Value/data element (output parameter, 32 bits; pointers are also possible).

putPosPtr: The address of the node element [▶ 243] (input parameter).

VAR_OUTPUT

```
VAR_OUTPUT
    bOk         : BOOL := FALSE;
    getValue    : DWORD := 0;
    getPosPtr   : POINTER TO T_LinkedListEntry := 0;
END_VAR
```

bOk: Result of the last action called. Returns TRUE if a new node element could be added, removed, or found in the list. FALSE is returned if the node element that was searched for could not be found, the list is empty, or has overflowed (no more free node elements).

getValue: Value/data element (output parameter, 32 bits; pointers are also possible).

getPosPtr: The address of the node element [▶ 243] (output parameter).

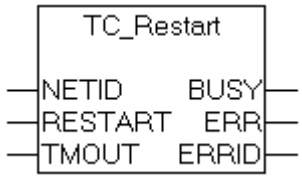
Example:

See: Example: Linked list (FB_LinkedListCtrl). [▶ 265]

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcUtilities.Lib

4.47 RestartTwinCAT



The function block "TC_Restart" can be used to restart the TwinCAT system. The function corresponds to the Restart command on the TwinCAT system menu (on the right of the Windows taskbar). Restarting the TwinCAT system involves the TwinCAT system first being stopped, and then immediately started again. Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    RESTART     :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the TwinCAT system is to be restarted. If the restart is to take place on the local computer, an empty string can be entered.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

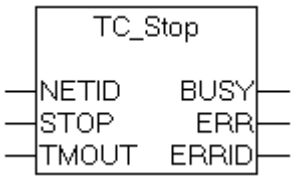
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.48 TC_Stop



The function block "TC_Stop" can be used to stop the TwinCAT system. The function corresponds to the Stop command on the TwinCAT system menu (on the right of the Windows taskbar).

Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID         :T_AmsNetId;
    STOP          :BOOL;
    TMOUT         :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the TwinCAT system is to be stopped. If the TwinCAT system to be stopped is on the local computer, an empty string can be entered.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

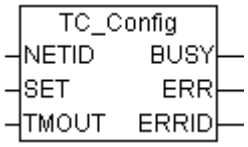
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.49 TC_Config



A TwinCAT system in RUN mode (green TwinCAT system icon) can be switched to CONFIG mode (blue TwinCAT system icon) via the function block "TC_Config". If the system is already in CONFIG mode, it is first switched to STOP mode (red TwinCAT system icon) and then to CONFIG mode. Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID        :T_AmsNetId;
    SET          :BOOL;
    TMOUT        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: This parameter can be used to specify the AmsNetId of the TwinCAT computer that is to be switched to CONFIG mode. If it is to be run on the local TwinCAT computer, an empty string can be entered.

SET: The block is activated by a rising edge at this input.

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

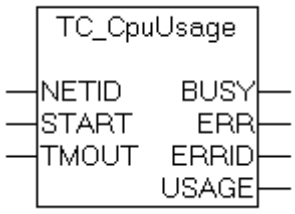
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.50 TC_CpuUsage



The "TC_CpuUsage" function block allows the current CPU loading of a TwinCAT system to be determined. This function corresponds to the display of CPU loading in the TwinCAT system menu under the real-time settings. Internally, an instance of the ADSREAD function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer whose CPU loading is to be determined If it is to be run on the local computer, an empty string can be entered.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

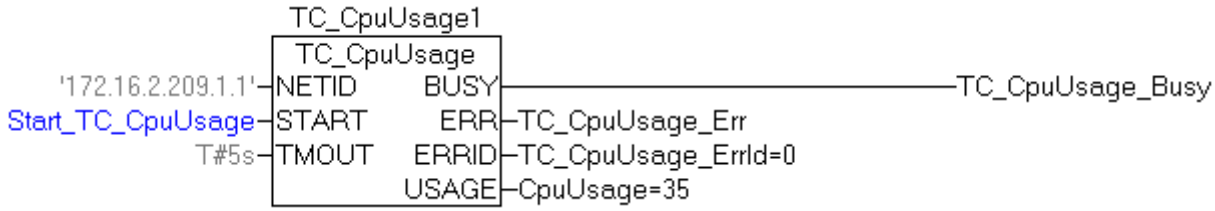
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

USAGE: The current CPU loading of a TwinCAT system in %.

Example of a call in FBD

```
TC_CpuUsage1          : TC_CpuUsage;
Start_TC_CpuUsage           : BOOL;
TC_CpuUsage_Busy         : BOOL;
TC_CpuUsage_Err      : BOOL;
TC_CpuUsage_ErrId           : UDINT;
CpuUsage             : UDINT;
```

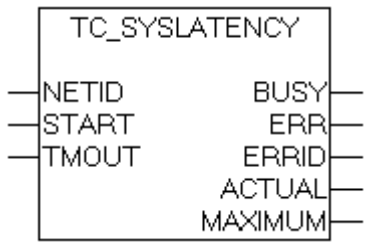


In the example the TwinCAT system is using 35% of the total available CPU computing time.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.51 TC_SysLatency



The "TC_SysLatency" function block allows the current and maximum latency time of a TwinCAT system to be determined. This function corresponds to the display of latency time in the TwinCAT system menu under the real-time settings. Internally, an instance of the ADSREAD function block is called.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer whose latency time is to be determined If it is to be run on the local computer, an empty string can be entered.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        :BOOL;
    ERR         :BOOL;
```

```
    ERRID      :UDINT;
    ACTUAL     :UDINT;
    MAXIMUM    :UDINT;
END_VAR
```

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

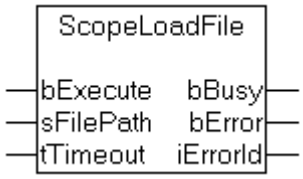
ACTUAL: The current latency time of a TwinCAT system in µs.

MAXIMUM: The maximal detected latency time of a TwinCAT system in µs.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 732	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.52 ScopeLoadFile



The function block "ScopeLoadFile" allows a Scope View Configuration Project (*.scp) to be loaded into the TwinCAT Scope View. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

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

sFilePath:: The full path with the name of the file that is to be loaded into the Scope View (e.g.: 'C:\TwinCAT\Scope\Achse1.scp').

tTimeout:: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

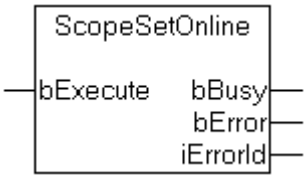
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.53 ScopeSetOnline



The function block "ScopeSetOnline" places the Scope View into the online state. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

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

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

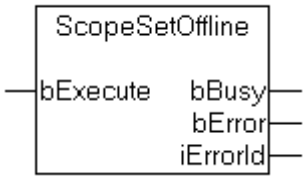
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.54 ScopeSetOffline



The function block "ScopeSetOffline" places the Scope View into the offline state. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

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

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

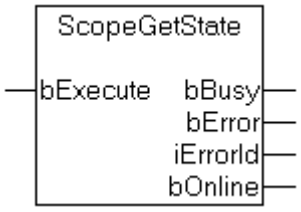
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.55 ScopeGetState



The function block "ScopeGetState" allows the state of the Scope View to be determined. If the Scope View is currently in the online state, the bOnline output is set. Internally, an instance of the ADSREAD function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

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

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    iErrorId    :UDINT;
    bOnline     :BOOL;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

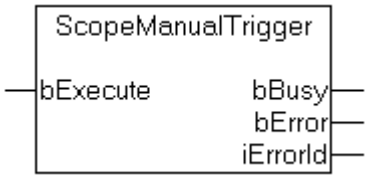
iErrorId: Supplies the ADS error number when the bError output is set.

bOnline: If this output is set, the Scope View is in the online state, otherwise it is offline.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.56 ScopeManualTrigger



The Scope View can be triggered manually with the function block "ScopeManualTrigger". Currently this function can only be used if only one view (such as Scope View 1) is active in the application. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

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

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

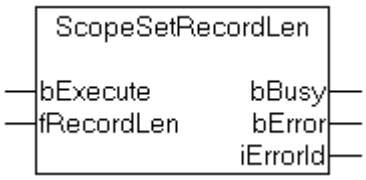
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.57 ScopeSetRecordLen



The function block "ScopeSetRecordLen" can be used to configure the Scope View recording time. Currently this function can only be used if only one view (such as Scope View 1) is active in the application. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    fRecordLen      :LREAL;
END_VAR
```

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

fRecordLen: The recording time in seconds for the Scope View that is to be configured.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

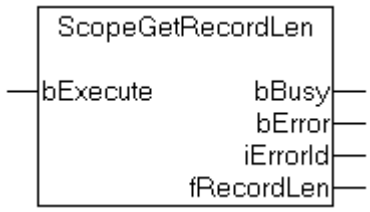
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.58 ScopeGetRecordLen



The function block "ScopeGetRecordLen" allows the recording time for the Scope View that is currently configured in TwinCAT Scope View to be determined. Currently this function can only be used if only one view (such as Scope View 1) is active in the application. Internally, an instance of the ADSREAD function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

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

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    iErrorId    :UDINT;
    fRecordLen  :LREAL;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

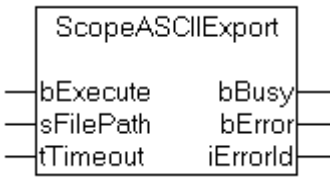
iErrorId: Supplies the ADS error number when the bError output is set.

fRecordLen: The recording time in seconds for the Scope View that is currently configured.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.59 ScopeASCIIExport



The function block "ScopeASCIIExport" allows the Scope View to be exported into an ASCII file. Currently this function can only be used if only one view (such as Scope View 1) is active in the application. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

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

sFilePath:: The full path with the name of the file into which the export is to be made (e.g.: 'C:\TwinCAT\Scope\Achse1.dat').

tTimeout:: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

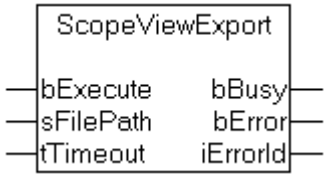
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.60 ScopeViewExport



The function block "ScopeViewExport" allows a Scope View to be exported into a file (*.svw). Currently this function can only be used if only one view (such as Scope View 1) is active in the application. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

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

sFilePath:: The full path with the name of the file into which the export is to be made (e.g.: 'C:\TwinCAT\Scope\Achse1.svw').

tTimeout:: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

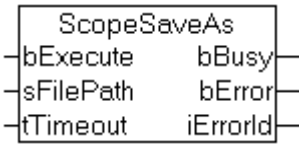
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.61 ScopeSaveAs



The function block "ScopeSaveAs" allows a Scope View Configuration Project (*.scp) to be saved under the specified file name. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

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

sFilePath:: The full path with the name of the file (e.g.: 'C:\TwinCAT\Scope\Achse1.scp').

tTimeout:: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

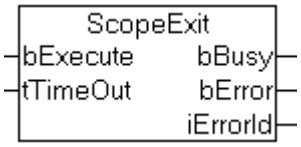
bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 740 TwinCAT v2.9.0 Build > 925	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.62 ScopeExit



The function block "ScopeExit" terminates one instance of TwinCAT Scope View. Internally, an instance of the ADSWRITE function block is called.

VAR_INPUT

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

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

tTimeout:: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

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

bBusy: When the function block is activated this output is set. It remains set until and acknowledgement is received.

bError: If an ADS error should occur during the transfer of the command, then this output is set once the bBusy output is reset.

iErrorId: Supplies the ADS error number when the bError output is set.

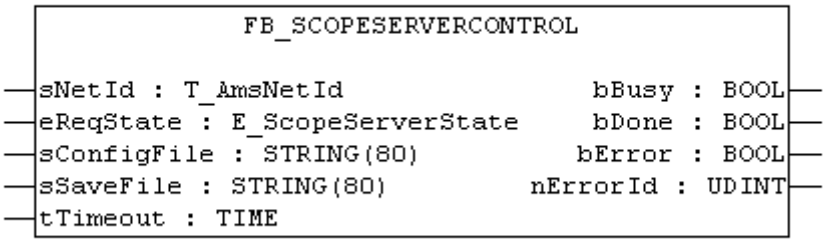
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 740 TwinCAT v2.9.0 Build > 925	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.63 FB_ScopeServerControl

[This documentation is preliminary and subject to modifications]

- As of library version 2.0.52
- Requires the installation of the TwinCAT Scope Server, which is delivered with the TwinCAT Scope View 2.



The function block "FB_ScopeServerControl" enables the PLC to collect data for subsequent display with TwinCAT Scope View 2.

VAR_INPUT

```
VAR_INPUT
   sNetId:      T_AmsNetId;
   eReqState:   E_ScopeServerState := SCOPE_SERVER_IDLE;
   sConfigFile: STRING;
   sSaveFile:   STRING;
   tTimeout:    TIME := DEFAULT_ADS_TIMEOUT;
ND_VAR
```

sNetId: Here a string containing the network address of the target TwinCAT Computer can be given. The string can also be empty for the local computer.

eReqState:: Requested Scope Server state [▶ 208].

sConfigFile:: The full path with the name of the configuration file (e.g.: 'C:\TwinCAT\TwinCATScopeView2\First.sv2').

sSaveFile: The full path with the name of the data file (e.g.: 'C:\TwinCAT\TwinCATScopeView2\First.svd').

tTimeout:: Maximum time allowed for the internal ADS commands

VAR_OUTPUT

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

bBusy: if the function block is activated, this output is set. It remains set until a feedback is received.

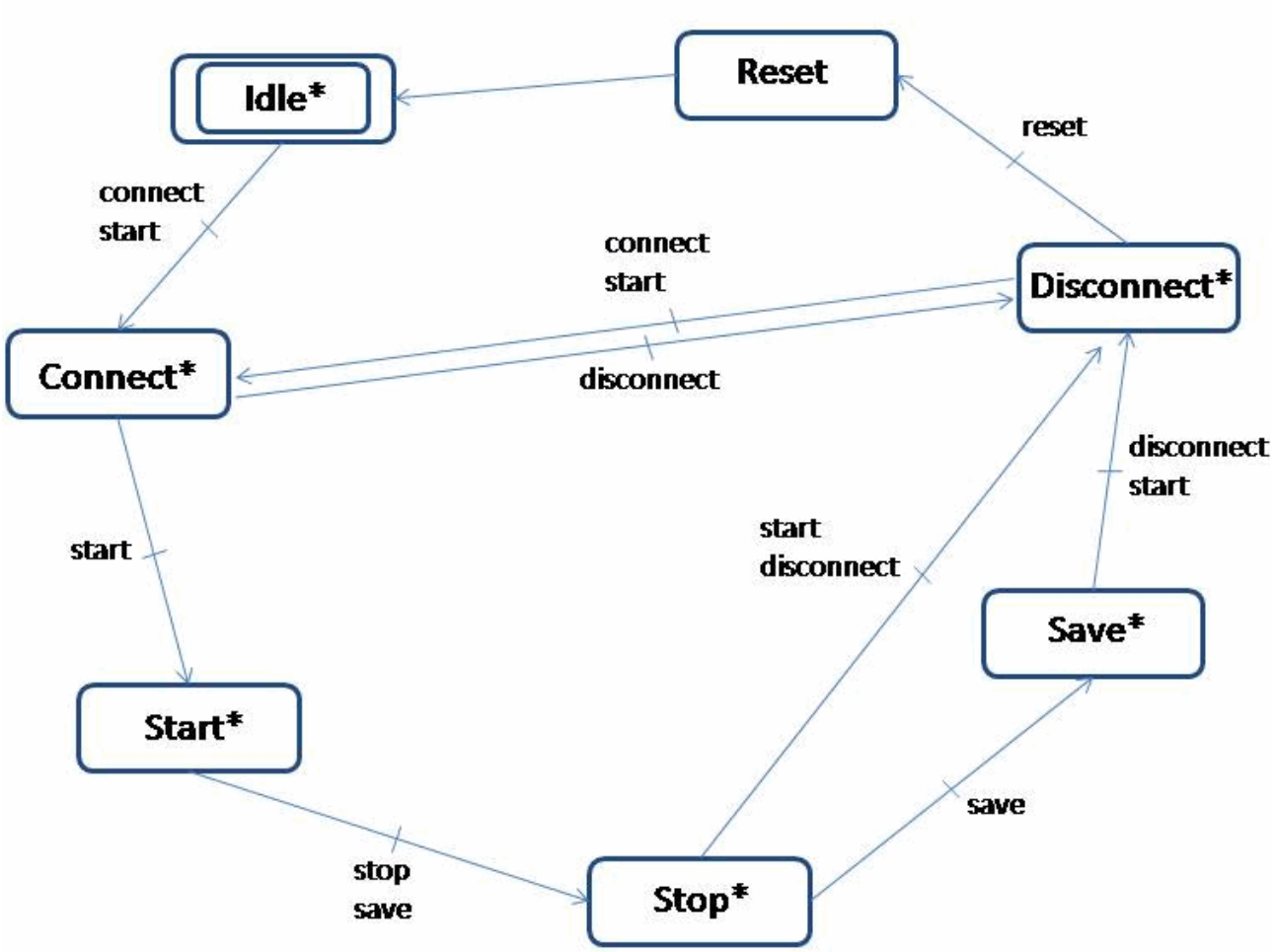
bDone: is set if the requested status was activated.

bError: if an ADS error should occur during the transmission of the command, this output is set after the *bBusy* output has been reset.

nErrorId: shows the error number if the error output is set. As a rule this can be an ADS error number or a specific error code [▶ 251] of this library.

Only one target system is permitted for the configuration file (*.sv2).

Status diagram:



*) resetting a state change is possible at any time

This status diagram shows the possible transitions for eReqState. If another state transition is requested, bError is set.

TYPE E_ScopeServerState

E_ScopeServerState

```
TYPE E_ScopeServerState
(
   SCOPE_SERVER_IDLE,
   SCOPE_SERVER_CONNECT,
   SCOPE_SERVER_START,
   SCOPE_SERVER_STOP,
   SCOPE_SERVER_SAVE,
   SCOPE_SERVER_DISCONNECT,
   SCOPE_SERVER_RESET
);
```

Example in ST:

Declaration part:

```
FUNCTION_BLOCK FB_ScopeServerSample
VAR_INPUT
   bExternalTriggerEvent: BOOL := FALSE;
END_VAR

VAR_OUTPUT
END_VAR

VAR
   fbScopeServerControl: FB_ScopeServerControl;
   eReqState: E_ScopeServerState := SCOPE_SERVER_IDLE;

   bBusy: BOOL := FALSE;
   bDone: BOOL := FALSE;
   bError: BOOL := FALSE;
   nErrorId: UDINT := 0;
   fbTimer: TON;
   bTriggerTimer: BOOL := FALSE;
   nState: UDINT := 0;
END_VAR
```

Implementation of FB_ScopeServerSample

```
CASE nState OF
0:

   eReqState := SCOPE_SERVER_START;
   nState := 10;

10:
   IF fbScopeServerControl.bDone AND bExternalTriggerEvent THEN
      bTriggerTimer := TRUE;
      nState := 20;
   END_IF

20:
   IF fbTimer.Q THEN
      eReqState := SCOPE_SERVER_SAVE;
      bTriggerTimer := FALSE;
      nState := 30;
   END_IF

30:
   IF fbScopeServerControl.bDone THEN
      eReqState := SCOPE_SERVER_DISCONNECT;
   END_IF
END_CASE


fbTimer(IN:=bTriggerTimer, PT:=t#10s);
fbScopeServerControl(
   sNetId:= '',
   eReqState:= eReqState,
   sConfigFile:= 'C:\twinCat\scope\test.sv2',
   sSaveFile:= 'C:\twinCat\scope\test.svd',
   tTimeout:= t#5s,
   bBusy=>bBusy,
   bDone=>bDone,
   bError=>bError,
   nErrorId=>nErrorId);
```

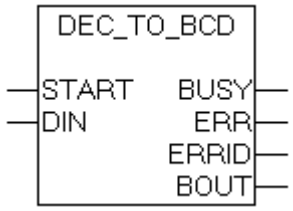
This sample should show, how a long-term data logging with the Scope Server could be handled.

Therefore an existing configuration (Test.sv2) is loaded. In this sample Test.sv2 was saved to run in Ring Buffer. So, data logging will not stopped until it is triggered by FB_ScopeServerControl. If there is an external trigger event (that could be an error event), a timer is started and 10 seconds later the data is saved in Test.svd. On that way the data file holds an information about what happens before and after the trigger event.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1329	PC or CX (x86,ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

4.64 DEC_TO_BCD



The "DEC_TO_BCD" function block allows decimal numbers to be converted to BCD format. The number to be converted is checked for admissibility of the values.

VAR_INPUT

```
VAR_INPUT
    START          : BOOL;
    DIN            : BYTE;
END_VAR
```

START: the function block is enabled via a positive edge at this input.

DIN: the decimal number to be converted.

VAR_OUTPUT

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

BUSY: this input is set at the start of the conversion procedure, and remains set until the conversion has been completed. Once the BUSY output has been reset, the BCD value is available at the BOUT output.

ERR: this variable is set to TRUE if an error occurs.

ERRID: error code.

BOUT: the converted variable in BCD format is available at this output if the process is successful.

Error Codes:

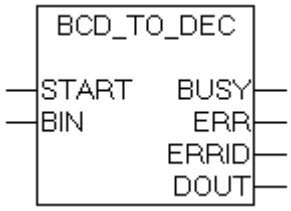
Error code	Error description
0	No error
0x00FF	The decimal value of the variable to be converted is not reliable

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;

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

4.65 BCD_TO_DEC



The "BCD_TO_DEC" function block allows numbers in BCD to be converted to decimal format. The BCD number to be converted is checked for the reliability of the values.

VAR_INPUT

```
VAR_INPUT
    START          :BOOL;
    BIN            :BYTE;
END_VAR
```

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

BIN: The BCD number to be converted.

VAR_OUTPUT

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

BUSY: This input is set at the start of the conversion procedure, and remains set until the conversion has been completed. Once the BUSY output has been reset, the decimal value is available at the DOUT output.

ERR: This variable is set to TRUE if an error occurs.

ERRID: Error code.

DOUT: The converted variable in decimal format is available at this output if the process is successful.

Error Codes:

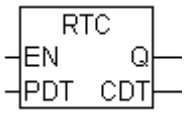
Error code	Error descriptions
0	no error
0x000F	Unreliable value in the low nibble of the BCD number
0x00F0	Unreliable value in the high nibble of the BCD number

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.66 RTC



The function block "RTC" (Real Time Clock) can be used to realize an internal software clock in the TwinCAT PLC. The clock must be initialized with a starting date and time. After the initialization the time and date are updated with each call of the function block. A CPU system clock is used to calculate the current time and date. The function block should be called in every PLC cycle, so that the current time can be calculated. At the output of the function block the current date and time is available in the common DATE_AND_TIME (DT) format. Multiple instances of the RTC function block can be created within one PLC program.

The way the system works means that the RTC time can differ from the reference time. The difference depends on the PLC's cycle time, the value of the basic system ticks, and on the hardware being used. In order to avoid larger deviations the RTC instance should be synchronized cyclically (e.g. with a radio clock or with the local Windows system time). The local Windows system time you can be synchronized with a reference time via the SNTP protocol.

VAR_INPUT

```
VAR_INPUT
    EN   : BOOL;
    PDT  : DATE_AND_TIME;
END_VAR
```

EN: The function block is re-initialized with a specified date and time by a rising edge at this input.
PDT: (Preset Date and Time) The initialization values for the function block's date and time. A rising edge at the EN input will cause the function block to adopt this value.

VAR_OUTPUT

```
VAR_OUTPUT
    Q    : BOOL;
    CDT  : DATE_AND_TIME;
END_VAR
```

Q: This output is set if the function block has been initialized at least once. If the output is set, the values for the date and time at the PDT output are valid.
CDT: (Current Date and Time) The current date and time from the RTC instance. The CDT output is only updated when the function block is called. For this reason, instances of the function block should be called once in each PLC cycle.

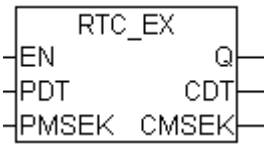
Example:

See: Example: Software clocks (RTC, RTC_EX, RTC_EX2) [▶ 273].

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 and higher TwinCAT v2.10.0 Build > 1232 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.67 RTC_EX



The "RTC_EX" (Extended Real Time Clock) function block allows an internal Software clock to be implemented in TwinCAT PLC. The clock must be initialized with a starting date and time. After the initialization the time and date are updated with each call of the function block. A CPU system clock is used to calculate the current time and date. The function block should be called in every PLC cycle, so that the current time can be calculated. At the output of the function block the current date and time is available in the common DATE_AND_TIME (DT) format. In contrast to the RTC [▶ 212] function block, RTC_EX has an accuracy of one millisecond. Multiple instances of the RTC_EX function block can be created within one PLC program.

The way the system works means that the RTC_EX time can differ from the reference time. The difference depends on the PLC's cycle time, the value of the basic system ticks, and on the hardware being used. In order to avoid larger deviations the RTC_EX instance should be synchronized cyclically (e.g. with a radio clock or with the local Windows system time). The local Windows system time you can be synchronized with a reference time via the SNTP protocol.

VAR_INPUT

```
VAR_INPUT
   EN     : BOOL;
   PDT    : DATE_AND_TIME;
   PMSEK  : DWORD;
END_VAR
```

EN: The RTC_EX function block is re-initialised with a specified date, time and millisecond by a rising edge at this input.
PDT: (Preset Date and Time) The initialisation values for the function block's date and time. A rising edge at the EN input will cause the function block to adopt this value.
PMSEK: (Preset Milliseconds) The initialisation value for the milliseconds. A rising edge at the EN input will cause the function block to adopt this value.

VAR_OUTPUT

```
VAR_OUTPUT
   Q      : BOOL;
   CDT    : DATE_AND_TIME;
   CMSEK  : DWORD;
END_VAR
```

Q: This output is set if the function block has been initialized at least once. If the output is set, the values for the date, time and milliseconds at the PDT and CMSEK outputs are valid.
CDT: (Current Date and Time) The current date and time from the RTC instance. The CDT output is only updated when the function block is called. For this reason, instances of the function block should be called once in each PLC cycle.
CMSEK: (Current Milliseconds) The milliseconds output.

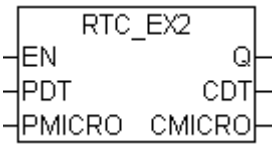
Example:

See: Example: Software clocks (RTC, RTC_EX, RTC_EX2) [▶ 273].

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 and higher TwinCAT v2.10.0 Build > 1232 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.68 RTC_EX2



The "RTC_EX2" (Extended Real Time Clock) function block allows an internal Software clock to be implemented in TwinCAT PLC. The clock must be initialized with a starting date and time. After the initialization the time and date are updated with each call of the function block. A CPU system clock is used to calculate the current time and date. The function block should be called in every PLC cycle, so that the current time can be calculated. At the output of the function block the current date and time is available in the Windows system time format. In contrast to the RTC [▶ 212] function block, RTC_EX2 has an accuracy of one microsecond. Multiple instances of the RTC_EX2 function block can be created within one PLC program.

The way the system works means that the RTC_EX2 time can differ from the reference time. The difference depends on the PLC's cycle time, the value of the basic system ticks, and on the hardware being used. In order to avoid larger deviations the RTC_EX2 instance should be synchronized cyclically (e.g. with a radio clock or with the local Windows system time). The local Windows system time you can be synchronized with a reference time via the SNTP protocol.

VAR_INPUT

```
VAR_INPUT
   EN       : BOOL;
   PDT      : TIMESTRUCT;
   PMICRO   : DWORD;
END_VAR
```

EN: The RTC_EX2 function block is re-initialized with a specified date, time and millisecond by a rising edge at this input.
PDT: (Preset Date and Time [▶ 230]) The initialization values for the function block's date and time. A rising edge at the EN input will cause the function block to adopt this value.
PMICRO: (Preset Microseconds) The initialization value for the microseconds. A rising edge at the EN input will cause the function block to adopt this value.

VAR_OUTPUT

```
VAR_OUTPUT
   Q                : BOOL;
   CDT              : TIMESTRUCT;
   CMICRO           : DWORD;
END_VAR
```

Q: This output is set if the function block has been initialised at least once. If the output is set, the values for the date, time and milliseconds at the PDT and CMSEK outputs are valid.
CDT: (Current Date and Time) The current date and time from the RTC_EX2 instance. The CDT output is only updated when the function block is called. For this reason, instances of the function block should be called once in each PLC cycle.
CMICRO: (Current Microseconds) The microseconds output.

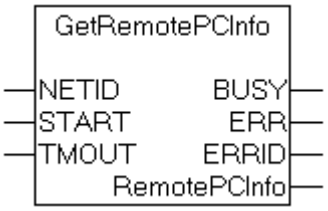
Example:

See: Example: Software clocks (RTC, RTC_EX, RTC_EX2) [▶ 273].

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 and higher TwinCAT v2.10.0 Build > 1232 and higher	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.69 GetRemotePCInfo



The function block "GetRemotePCInfo" permits information about configured remote PCs in the TwinCAT Router to be read. After successful execution, the structure "RemotePCInfo" contains the NetIds and names of the remote PCs as strings, in the same sequence as that in which they are stored in the TwinCAT Router. Internally, an instance of the ADSREAD function block is called. The function block allows router information relating to either a local or to a remote TwinCAT system to be read.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: The AmsNetId of the TwinCAT computer whose router information about configured remote PCs is to be read can be given here. If the remote PCs of the local TwinCAT system are to be found, it is also possible to enter an empty string.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

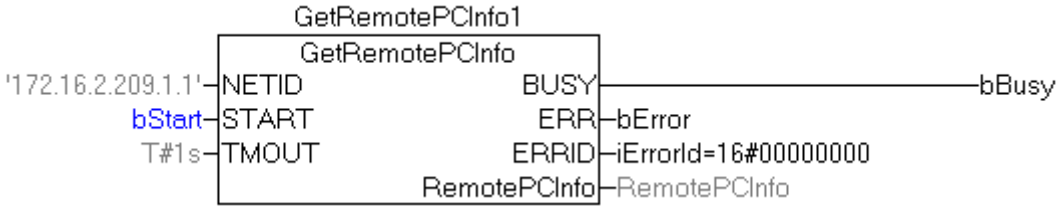
ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

RemotePCInfo: Structure [▶ 232] containing information about the configured remote PCs.

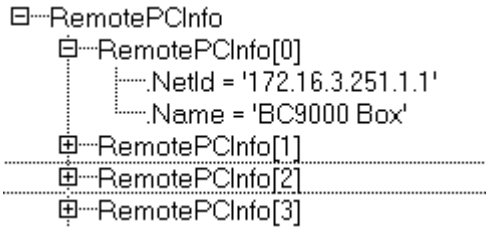
Example of a call in FBD:

```
PROGRAM MAIN
VAR
    GetRemotePCInfo1    : GetRemotePCInfo;
    RemotePCInfo        : REMOTEPCINFOSTRUCT;

    bBusy       : BOOL;
    bError      : BOOL;
    iErrorId        : UDINT;
    bStart      : BOOL;
END_VAR
```



Online View:

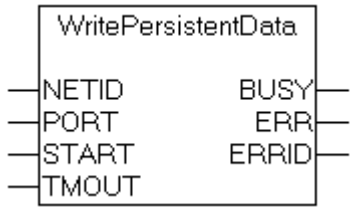
The NetIds and names of the configured remote PCs.



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.70 WritePersistentData



If persistent variables are defined in one of the PLC's runtime systems, then their current values are stored in a file in the TwinCAT\Boot directory when the TwinCAT system stops (after the last PLC cycle). A file is created for every runtime system that is configured. The next time the system starts, this file is read, and the persistent variables in the runtime system are initialized with the values from the file. Using the function block "WritePersistentData" it is possible to initiate storage of the persistent data from the PLC program. The PORT parameter specifies the runtime system whose persistent data is to be stored. For additional info see: Writing of persistent data: System behavior [▶ 277]. Internally, an instance of the ADSWRTCTRL function block is called.

VAR_INPUT

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

NETID: It is possible here to provide the AmsNetId of the TwinCAT computer on which the ADS command is to be executed. If it is to be run on the local computer, an empty string can be entered.

PORT: Contains the ADS port number of the PLC run-time system whose persistent data is to be stored. The first PLC run-time system, for example, has port number 801, and the second has port number 811.

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

TMOUT: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

BUSY: When the function block is activated this output is set. It remains set until and acknowledgement is received.

ERR: If an ADS error should occur during the transfer of the command, then this output is set once the BUSY output is reset.

ERRID: Supplies the ADS error number when the ERR output is set.

PLC Example Program in ST:

```
PROGRAM MAIN
VAR
    bStart                : BOOL;
    bError                : BOOL;
    bBusy                 : BOOL;
    nErrorId              : UDINT;
    fbWritePersistentData : WritePersistentData;
    fbR_Trig              : R_TRIG;
END_VAR

VAR PERSISTENT
    perA        : INT;
    perB        : BOOL;
    perC        : BYTE;
    perD        : STRING;
    perE        : ARRAY[0..10] OF INT;
    perF        : ARRAY[0..10] OF UDINT;
END_VAR
```

```
fbR_Trig( CLK:=bStart );

IF fbR_Trig.Q THEN

    perA := 24443;
```

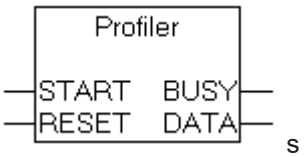
```
    perB := TRUE;
    perC := 7;
    perD := 'Switch ON/OFF';
    perE[ 0 ] := 1;
    perE[ 10 ] := 11;
    perF[ 0 ] := 263;
    perF[ 10 ] := 23323;

    fbWritePersistentData(NETID:='', PORT:=801, START:=bStart, TMOUT:=T#1s );
ELSE
    fbWritePersistentData( START:=FALSE);
END_IF;

bBusy := fbWritePersistentData.BUSY;
bError := fbWritePersistentData.ERR;
nErrorId := fbWritePersistentData.ERRID;
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.71 Profiler

This functionality is not available in the PLC under Windows CE!



The "Profiler" function block can be used to allow the execution time of PLC code to be measured. Internally, an instance of the GETCPUACCOUNT function block is called. The measurement is started by a rising edge at the START input, and is stopped by a falling edge. The measurements are evaluated internally, and are then made available for further processing at the DATA output in a structure of type PROFILERSTRUCT [▶ 231]. In addition to the current, minimum and maximum execution times, the function block calculates the mean execution time for the last 10 measurements. The number of averaged measured values can be configured via the global variable MAX_AVERAGE_MEASURES [▶ 250] between 2 and 100. The times measured are given in microseconds. The output variable DATA.MasureCycle [▶ 231] provides information about the number of measurements that have already been carried out. In order to measure the execution time for a specific segment of the PLC program the measurement must be started by a rising edge at the START input when the segment to be measured starts, and stopped by a falling edge at the START input at the end of the segment. All values at the DATA output can be reset if a rising edge is generated at the RESET input at the same time as the rising edge at START. The old measured values are then reset when a new measurement starts and are recalculated from the subsequent calls of the function block.

The times measured can differ from the actual values, since a certain amount of time is needed just for the call of the GETCPUACCOUNT function block. This time depends on the particular computer, and is included in the times that are found.

VAR_INPUT

```
VAR_INPUT
    START         :BOOL;
    RESET         :BOOL;
END_VAR
```

START: A rising edge at this input starts the measurement of the execution time. A falling edge at this input stops the measurement, and causes the current, minimum, maximum and mean execution times to be recalculated. The variable DATA.MeasureCycle [▶ 231] is incremented at the same time.

RESET: A rising edge at the this input and simultaneous rising edge at START input will reset the variables at the DATA output. The old values for the current, minimum, maximum and mean execution times are reset, and are re-calculated for following measurements.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY          :BOOL;
    DATA          :PROFILERSTRUCT;
END_VAR
```

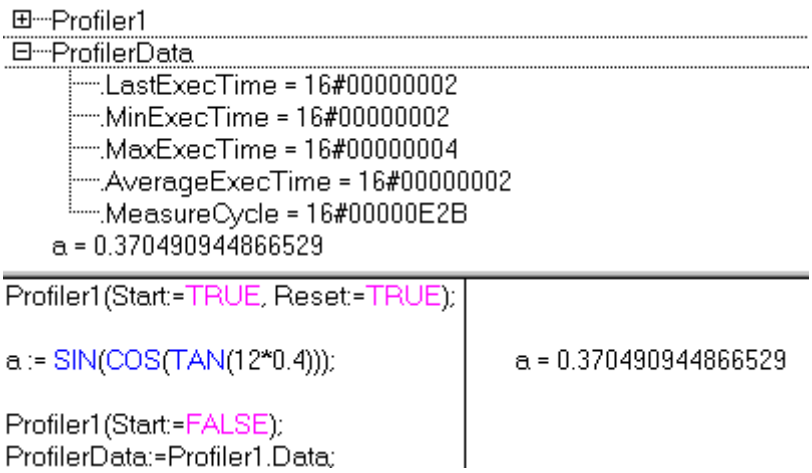
BUSY: This input is set at the start of the measuring procedure, and remains set until the time measurement has been completed. Once the BUSY output has been reset, the latest times are available at the DATA output.

DATA: structure of type PROFILERSTRUCT [▶ 231] with the measured times [in µs].

Example of a call in ST:

```
PROGRAM ProfilerTest_ST
VAR
    PROFILER1     :PROFILER;
    ProfilerData  :PROFILERSTRUCT;
    a         :LREAL;
END_VAR
```

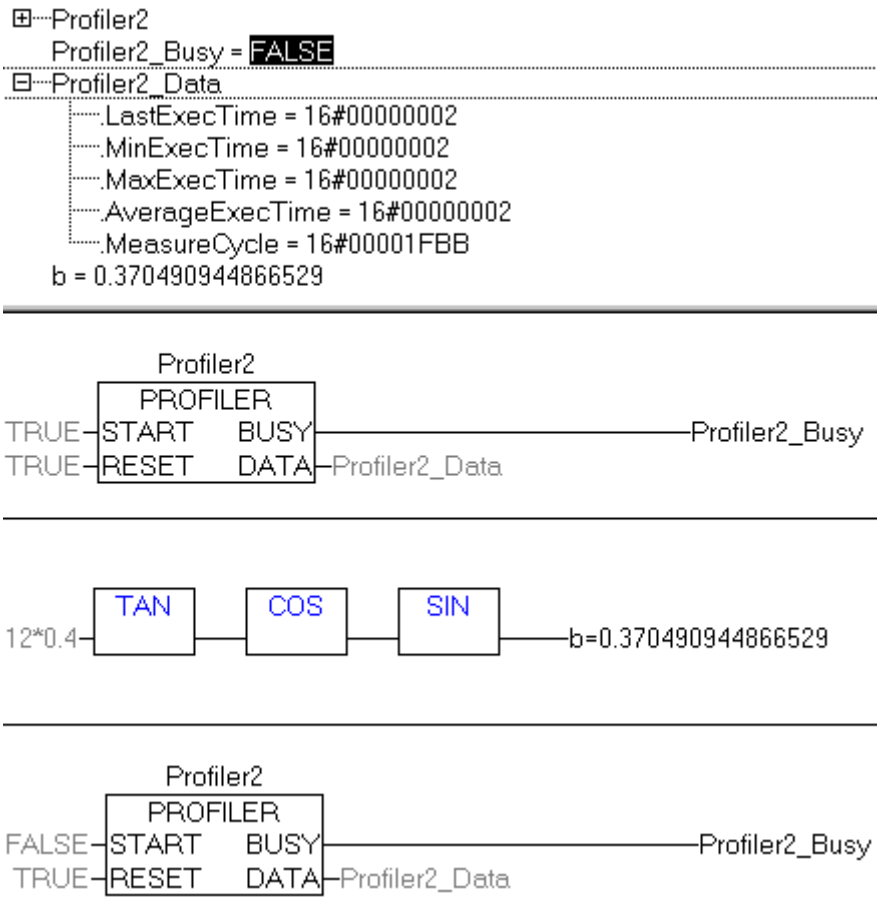
Online display of the measured times:



Example of a call in FBD:

```
PROGRAM ProfilerTest_FUP
VAR
    Profiler2     :PROFILER;
    Profiler2_Busy    :BOOL;
    Profiler2_Data    :PROFILERSTRUCT;
    b         :LREAL;
END_VAR
```

Online display of the measured times:



Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.72 DCF77_TIME

The "DCF77_TIME" function block can be used to decode the DCF-77 radio clock signal.

This function block is replaced by the new "DCF77_TIME_EX [▶ 224]" function block.

A rising edge at the RUN input starts the decoding process, which continues as long as the RUN input remains set. In the worst case, the function block requires a maximum of 1 minute to synchronize itself, plus a further minute to decode the data for the following minute. During this time, the missing 59th second marker is waited for. Internally the function block is sampling the DCF-77 signal. In order to be able to sample the edges without error the function block should be called once in each PLC cycle. Satisfactory results can be obtained with a cycle time of <= 25 ms. In case of a missing or faulty DCF-77 signal, the ERR output is set to TRUE and a corresponding error code is set at the ERRID output. The ERR and ERRID outputs are reset the next time a correct signal is received. Some receivers provide an inverted DCF-77 signal. In such cases the signal must first be inverted before being passed to the DCF_PULSE input. When operating without errors, the current time is updated at the CDT output every minute. Thereby the READY output is set to TRUE for one cycle of the PLC at the zeroth second. At this time the DCF-77 time at the CDT output is valid, and can be evaluated by the PLC program. The READY output is only set if no error was detected in the data for the following minute. The transferred parity bits are used for error detection. In the event of poor reception conditions, 100% error-free detection cannot be guaranteed. I.e. with two faulty (inverted) bits, the function block cannot detect an error and also sets the READY output to TRUE. In order to obtain reliable time information additional safeguards have to be implemented, e.g. redundancy analysis of the time information in consecutive minutes.

From TwinCAT v2.10 Build > 1340 and TwinCAT v2.11 Build > 1542 a simple plausibility check of two consecutive telegrams was implemented in the DCF77_TIME function block. This functionality can be activated via a global Boolean variable for all instances of the DCF77_TIME block. When the plausibility check is activated the first synchronization is extended by a further minute to a maximum of 3 minutes.

```
GLOBAL_DCF77_SEQUENCE_CHECK : BOOL := FALSE;
(* TRUE = Enable plausibility check (two telegrams are checked), FALSE = Disable check *)
```

Errors that occur during reception are registered by the function block. The ERRCNT output is an error counter. This counter indicates the number of errors that have occurred since the last correctly received signal. The counter is reset the next time a signal is correctly received.

Time code

During each minute, the numbers that encode the year, month, day, day of the week, hour and minute are transmitted in BCD format through pulse modulation of the second marks. The transmitted information always describes the subsequent minute. A second marker is transmitted each second. A second marker with a duration of 0.1 s represents a binary zero, while a duration of 0.2 s represents binary one. The information is extended with 3 check bits. No second marker is transmitted for the 59th second, and a receiver can use this "gap" to synchronize itself.

From TwinCAT v2.10 Build > 1340 and TwinCAT v2.11 Build > 1542 the length of the short and long pulse signal can also be configured via a global variable. If the signal is poor the pulse widths are smaller. The receiver specification usually contains information about the minimum and maximum pulse for the two logic signals, with the higher value expected for higher field strengths and the lower value for low field strengths or in the event of interference. Problems may also occur near the sender (where the field strength is very large) if the pulse width of the logic zero becomes excessive. For this reason a fixed limit is set for differentiating between zero and one, depending on the receiver specification. **Check the specification of the receiver used and configure the pulse length accordingly.**

```
GLOBAL_DCF77_PULSE_SPLIT : TIME := T#140ms;(* 0 == pulse < 140ms, 1 == pulse > 140 *)
```

E.g.: in the specification of Atmel T4227 (Time Code Receiver) the following pulse length is given:
100ms pulse (zero): min: 70ms, typical: 95ms, max: **130ms**
200ms pulse (one): min **170ms**, typical 195ms, max 235ms
For this IC a limit value of 150ms would be optimal = 130 + ((170ms - 130ms) / 2).

If the configured limit value for the pulse length is too small, short pulses are detected as long. Conversely, if the configured limit value is too small, long pulses are detected as short. If the checksum is correct, the receiver cannot detect these errors. In the first case the receiver may supply times that are in future range, in the second case the times may be in the past.

VAR_INPUT

```
VAR_INPUT
    DCF_PULSE    :BOOL;
    RUN          :BOOL;
END_VAR
```

DCF_PULSE: the DCF-77 signal.
RUN: a rising edge at this input initializes the function block and starts decoding the DCF-77 signal. If this input is reset, the decoding process is stopped.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY         :BOOL;
    ERR          :BOOL;
    ERRID        :UDINT;
    ERRCNT       :UDINT;
    READY        :BOOL;
    CDT          :DATE_AND_TIME;
END_VAR
```

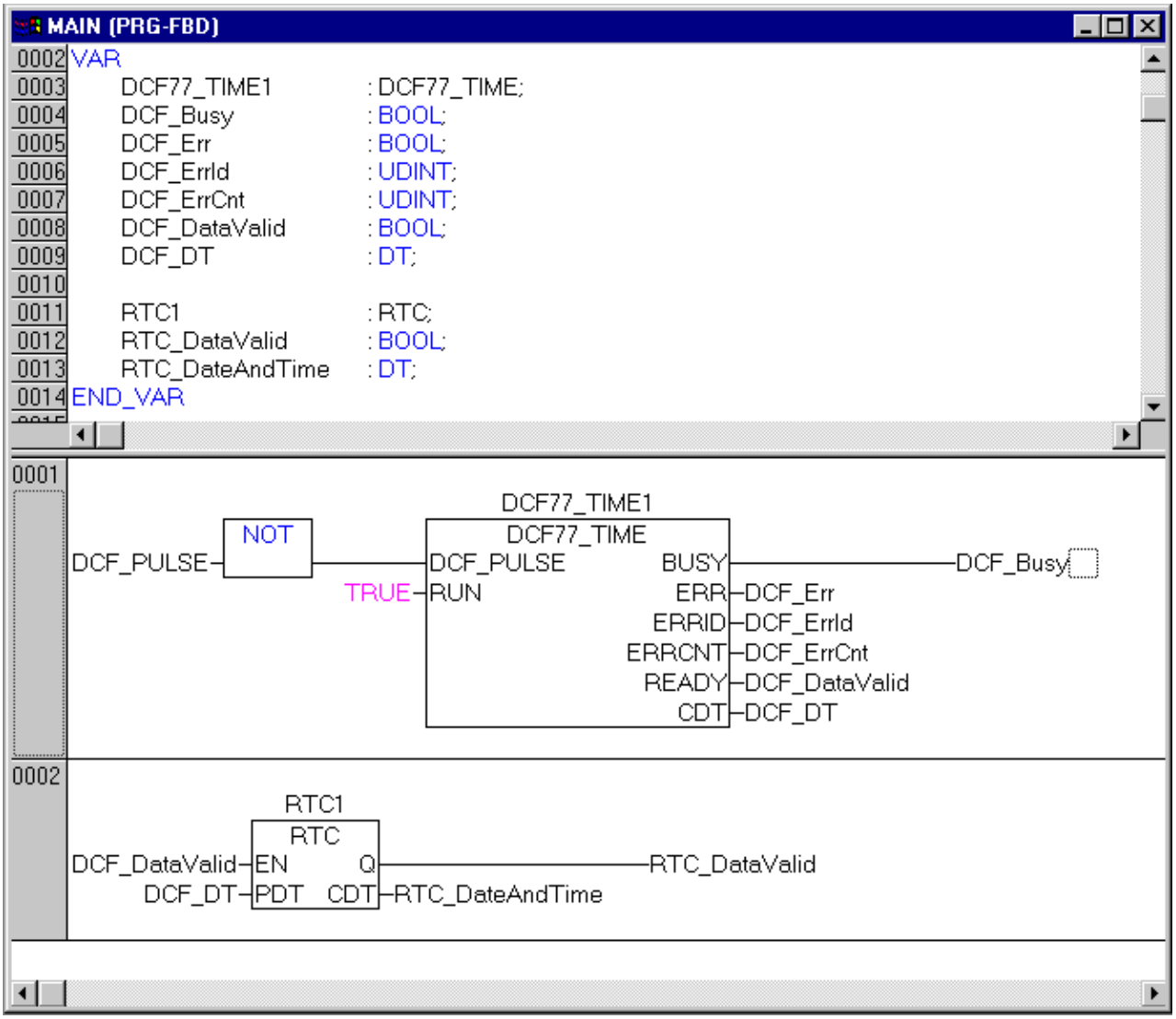
BUSY: when the function block is activated, this output is set.
ERR: if an error occurred during decoding, this output is set.
ERRID: supplies the error number when the ERR output is set.
ERRCNT: number of errors that have occurred since the last error-free reception.
READY: if this output is set, then the data at the CDT output are valid.
CDT: the DCF-77 time in DATE_AND_TIME format.

Error description:

Error Codes	Error description
0	No error
0x100	Timeout error. Possibly no DCF-77 signal detected.
0x200	Parity error. Incorrect bits were detected in the received data.
0x300	Faulty data received. Since the parity check can only detect one incorrect bit, the received data are checked again for validity (this error code will be generated, for instance, if month = 13).
0x400	The last decoding cycle was too long. This error can occur when reception is poor (not enough second markers were received).
0x500	The last decoding cycle was too short. This error can occur when reception is poor (additional edges were received).

Example of a call in FBD

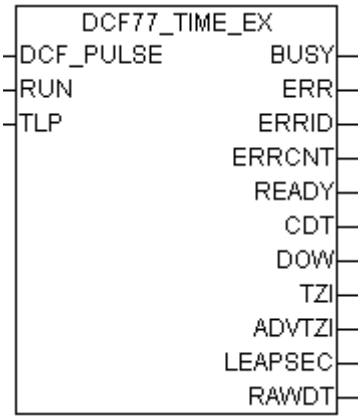
In the example application, the Real Time Clock is synchronized with the radio time in case of an error-free reception.



Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.10.0 Build > 1313 and higher	PC or CX (x86, ARM)	TcUtilities.Lib

4.73 DCF77_TIME_EX



The "DCF77_TIME_EX" function block can be used to decode the DCF-77 radio clock signal. In contrast to the DCF77_TIME [▶ 221] function block, this block checks two consecutive telegrams for plausibility as standard.

A rising edge at the RUN input starts the decoding process, which continues as long as the RUN input remains set. In the worst case synchronisation of the function block takes up to one minute and two further minutes for decoding data for the next minute. It is waiting, during that time, for the missing 59th second marker. Internally the function block is sampling the DCF-77 signal. In order to ensure error-free scanning of the edges, the function block should be called once in each PLC cycle. Satisfactory results can be achieved with a cycle time of <= 25ms. If the DCF-77 signal is absent or faulty, the ERR output is set TRUE, and a corresponding error code is set at the ERRID output. The ERR and ERRID outputs are reset the next time a correct signal is received. Some receivers provide an inverted DCF-77 signal. In such cases the signal must first be inverted before being passed to the DCF_PULSE input. When operating without errors, the current time is updated at the CDT output every minute. When this is happening, the READY output is set to TRUE for one PLC cycle in the first second (second zero). At this time the DCF-77 time at the CDT output is valid, and can be evaluated by the PLC program. The READY output is only set if it was possible to receive the data for the coming minute without errors. The transferred parity bits are used for fault detection, and the last two telegrams are checked for plausibility. When reception is poor 100% error-free identification cannot be guaranteed, in the event of two defective (inverse) bits in two subsequent telegrams the function block cannot detect an error and will set the READY output to TRUE. Due to the plausibility check the probability of the appropriate bits becoming distorted, thereby preventing detection of such an error, is quite small.

Errors that occur during reception are registered by the function block. The ERRCNT output is an error counter. This counter indicates the number of errors that have occurred since the last correctly received signal. The counter is reset the next time a signal is correctly received.

Time code

During each minute, the numbers that encode the year, month, day, day of the week, hour and minute are transmitted in BCD format through pulse modulation of the second marks. The transmitted information always describes the subsequent minute. A second marker is transmitted each second. A second marker with a duration of 0.1 s represents a binary zero, while a duration of 0.2 s represents binary one. The information is extended with 3 check bits. No second marker is transmitted for the 59th second, and a receiver can use this "gap" to synchronise itself.

VAR_INPUT

```
VAR_INPUT
    DCF_PULSE   :BOOL;
    RUN         :BOOL;
    TLP         :TIME := 140ms;
END_VAR
```

DCF_PULSE: The DCF-77 signal.
RUN: A rising edge at this input initialises the function block and starts decoding the DCF-77 signal. If this input is reset, the decoding process is stopped.
TLP: Via this input a fixed limit is set for differentiating between zero and one, depending on the recipient specification. If the signal is poor the pulse widths are smaller. The receiver specifications usually contain information about the minimum and maximum pulse for the two logic signals, with the higher value expected for higher field strengths and the lower value for low field strengths or in the event of interference. Problems may also occur near the sender (where the field strength is very large) if the pulse width of the logic zero becomes excessive. **Check the specification of the receiver used and configure the impulse length accordingly.**

E.g.: the Atmel T4227 specification (time code receive) contains the following pulse length specification:
100 ms pulses (zero): min: 70 ms, typical: 95 ms, max: **130 ms**
200 ms pulses (one): min **170 ms**, typical 195 ms, max 235 ms
For this IC a limit value of 150 ms would be ideal (130 + ((170 ms – 130 ms) / 2)).

If the configured limit value for the impulse length is too small, short impulses are detected as long. Conversely, if the configured limit value is too small, long impulses are detected as short. If the checksum is correct, the receiver cannot detect these errors. In the first case the receiver may supply times that are in future range, in the second case the times may be in the past.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY    :BOOL;
    ERR     :BOOL;
    ERRID   :UDINT;
    ERRCNT  :UDINT;
    READY   :BOOL;
    CDT     :DATE_AND_TIME;
    DOW     :BYTE(1..7);(* ISO 8601 day of week: 1 = Monday.. 7 = Sunday *)
    TZI     :E_TimeZoneID;(* time zone information *)
    ADVTZI  :BOOL;(* MEZ->MESZ or MESZ->MEZ time change notification *)
    LEAPSEC :BOOL;(* TRUE = Leap second *)
    RAWDT   :ARRAY[0..60] OF BOOL;(* Raw decoded data bits *)
END_VAR
```

BUSY: This output is set when the function block is activated.
ERR: This output is set when an error occurs in the decoding.
ERRID: Supplies the error number when the ERR output is set.
ERRCNT: The number of errors that have occurred since the last correctly received signal.
READY: If this output is set, the data at the CDT output is valid.
CDT: The DCF-77 time in DATE_AND_TIME format.
DOW: day of the week according to ISO 8601: 1 = Monday... 7 = Sunday.
TZI: time zone information (daylight-saving time/standard time).
ADVTZI: daylight-saving time/standard time changeover notification, e.g. CET -> CEST or CEST -> CET. The changeover between CEST/CET occurs at the end of this hour (see telegram examples).
LEAPSEC: leap second notification. A leap second is added at the end of this hour (see telegram examples).
RAWDT: last decoded (raw) bit information. Please ensure that only the parity bits of the time information are checked. The parity bits of the weather data are not analysed.

Error descriptions:

Error Codes	Error descriptions
0	No error
0x100	Timeout error. Possibly because no DCF-77 signal is detected.
0x200	Parity error. Incorrect bits were detected in the received data.
0x300	Incorrect data was received. Since the parity check can only detect one incorrect bit, the received data is checked again for validity (this error code will be generated, for instance, if month = 13).
0x400	The last decoding cycle was too long. This error can occur when reception is poor (not enough second markers were received).

Error Codes	Error descriptions
0x500	The last decoding cycle was too short. This error can occur when reception is poor (additional edges were received).

Telegram examples:

CEST -> CET (daylight-saving time -> standard time)

'0 01110100100111 011001 00011011 0100001 011001 111 00001 000100000': Sunday, 26.10.08 02:58:00, TZI = eTimeZoneID_Daylight, ADVTZI = TRUE
'0 11110001101110 011001 10011010 0100001 011001 111 00001 000100000': Sunday, 26.10.08 02:59:00, TZI = eTimeZoneID_Daylight, ADVTZI = TRUE
'0 01000001001110 010101 00000000 0100001 011001 111 00001 000100000': Sunday, 26.10.08 02:00:00, TZI = eTimeZoneID_Standard, ADVTZI = TRUE
'0 01111110100000 000101 10000001 0100001 011001 111 00001 000100000': Sunday, 26.10.08 02:01:00, TZI = eTimeZoneID_Standard, ADVTZI = FALSE

CET -> CEST (standard time -> daylight-saving time)

'0 01000110111010 010101 00011011 1000001 000011 111 11000 000100000': Sunday, 30.03.08 01:58:00, TZI = eTimeZoneID_Standard, ADVTZI = TRUE
'0 01000010100111 010101 10011010 1000001 000011 111 11000 000100000': Sunday, 30.03.08 01:59:00, TZI = eTimeZoneID_Standard, ADVTZI = TRUE
'0 10000111100011 011001 00000000 1100000 000011 111 11000 000100000': Sunday, 30.03.08 03:00:00, TZI = eTimeZoneID_Daylight, ADVTZI = TRUE
'0 01010000010110 001001 10000001 1100000 000011 111 11000 000100000': Sunday, 30.03.08 03:01:00, TZI = eTimeZoneID_Daylight, ADVTZI = FALSE

Leap second

'0 10110000100001 000111 10011010 0000000 100000 001 10000 100100001': Thursday, 01.01.09 00:59:00, TZI = eTimeZoneID_Standard, LEAPSEC = TRUE
'0 11010010111000 000111 00000000 1000001 100000 001 10000 1001000010': Thursday, 01.01.09 01:00:00, TZI = eTimeZoneID_Standard, LEAPSEC = TRUE
'0 01000110011101 000101 10000001 1000001 100000 001 10000 100100001': Thursday, 01.01.09 01:01:00, TZI = eTimeZoneID_Standard, LEAPSEC = FALSE

: LEAPSEC bit;

: CET/CEST-Information;

: ADVTZI bit;

Example:

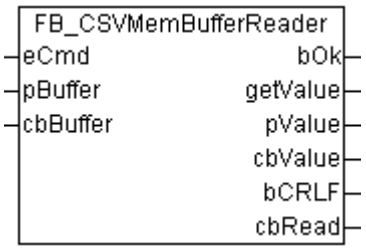
See description of the "DCF77_TIME [▶ 221]" function block.

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1340 and higher	PC or CX (x86, ARM)	TcUtilities.Lib

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1542 and higher		

4.74 FB_CSVMemBufferReader



This function block can be used to decompose/interpret data sets stored in an external buffer into individual data fields. The buffer data could first be read from a file with the aid of the function blocks for file access, for example. The function block reads the first or the next data field and returns its value either as a string at the *getValue* output or as an address/byte value at the *pValue*/*cbValue* output.

The data in the buffer must have a certain format to ensure that the function block can interpreted them correctly. CRLF (CR=Carriage Return, LF=Line Feed) must be used as a data set separator. The last data set must end with a CRLF. Individual data fields must be separated with the data field separator. The default data field separator is a semicolon. The separator can be changed from semicolon to comma via the global PLC variable **DEFAULT_CSV_FIELD_SEP**.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
END_VAR
```

eCmd: Control parameter [▶ 235] for the buffer component: eEnumCmd_First reads the first data field, eEnumCmd_Next reads the next data field. No other parameter values are used.

pBuffer: Address (pointer) for the source buffer variable. The address can be determined with the ADR operator. This buffer contains the data set/data field data to be read.

cbBuffer: The byte size of the data to be interpreted in the source buffer (data set/data field data). The buffer size may be much larger than the amount of data to be interpreted. Please enter the actual length of the data to be interpreted.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk         : BOOL;
    getValue    : T_MaxString := '';
    pValue      : DWORD := 0;
    cbValue     : UDINT := 0;
    bCRLF       : BOOL := FALSE;
    cbRead      : UDINT := 0;
END_VAR
```

bOk: TRUE = Success, FALSE = faulty data/faulty input parameters, or the end of the data was reached and no further data field could be read.

getValue: The last read data field as a string. For data fields without control characters and binary data this output returns the complete data field as a zero-terminated string. Data fields with control characters or binary data may lead to an incomplete string being returned at this output. In this case the outputs *pValue*/*cbValue* are used to access the last read data field.

pValue: Address (pointer) to the first data byte in the data field. Please note that empty void data fields are not terminated with zero (as is usual for a PLC string) and therefore have no data. In this case the address is zero.

cbValue: Data field length in bytes. Please note that empty void data fields are not terminated with zero (as is usual for a PLC string) and therefore have no data. The length is also zero in this case.

bCRLF: This output is set if the end of the data set was reached during the last read command. The last read data field belongs to the previous data set. The next data field belongs to a new data set.

cbRead: Number of successfully read/interpreted data bytes. This number may be greater than the data field length at the *cbValue* output. The length at the *cbRead* output includes the interpreted data field/data set separators.

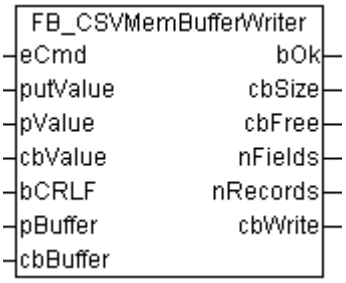
Example in ST:

Further information can be found here: Example: Writing/reading of CSV file [▶ 270].

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

4.75 FB_CSVMemBufferWriter



This function block can be used to generate data sets in an external buffer in CSV format from individual data fields. The content of the buffer can then be written into a file, e.g. with the aid of the function blocks for file access. The new data field can be transferred to the block either via the *putValue* variable (string) or via the optional *pValue* and *cbValue* variables, depending on whether data fields without control characters (string), data fields with control character or binary data are to be written into the data set. The function block can generate several data sets in the buffer until the maximum available buffer size is reached. The end of the data set (last data field in the current data set) is automatically appended to the data field if the bCRLF variable was set to TRUE during writing of the data field. The block automatically adds the data field separators. The default data field separator is a semicolon. The separator can be changed from semicolon to comma via the global PLC variable **DEFAULT_CSV_FIELD_SEP**.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    putValue    : T_MaxString := '';
    pValue      : DWORD := 0;(* OPTIONAL: Pointer to external buffer containing field value data. *)
    cbValue     : UDINT := 0;
(* OPTIONAL: Byte size of external buffer containing field value data. *)
    bCRLF       : BOOL := FALSE;(* TRUE = > Append end of record separator (CRLF) *)
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
END_VAR
```

eCmd: Control parameter [▶ 235] for the buffer component: eEnumCmd_First adds the first data field to the buffer, eEnumCmd_Next adds the next data field. No other parameter values are used.

putValue: A new data field as a string. This input must be an empty string if the optional parameters *pValue* and *cbValue* are used instead of this input.

pValue: Optional: Address of an external byte buffer containing the new data field. Together with the cbValue parameter this input can then be used to write a data field with control characters or binary data into the data set, for example. Control characters or binary data in the data field could truncate the *putValue* string at an undesired location and are therefore transferred as a byte buffer. This input must be zero if it is not used.

cbValue: Optional: Length of the data field data in the external byte buffer. This input must be zero if it is not used.

bCRLF: If this input set the new data field is terminated with a CRLF data set separator. Subsequent data fields belong to the new data set.

pBuffer: Address (pointer) for the target buffer variable. The address can be determined with the ADR operator. In this buffer the function block generates the data sets in CSV format.

cbBuffer: Maximum available size (in bytes) of the target buffer variables. The size can be determined with the SIZEOF operator.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     : BOOL;
    cbSize  : UDINT;
    cbFree  : UDINT;
    nFields : UDINT;
    nRecords: UDINT;
    cbWrite : UDINT;
END_VAR
```

bOk: TRUE = success, FALSE = buffer overflow or faulty input parameters.

cbSize: Current buffer fill status (number of data bytes created in the buffer).

cbFree: Number of free data bytes in the buffer.

nFields: Number of written data fields.

nRecords: Number of written data sets.

cbWrite: Number of last written data bytes (length of the last data field + any data set or data field separators).

Example in ST:

Further information can be found here: Example: Writing/reading of CSV file [▶ 270].

Requirements

Development environment	Target system type	PLC Libraries to include
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

5 Data types

5.1 TIMESTRUCT

```
TYPE TIMESTRUCT:
STRUCT
   wYear         : WORD;
   wMonth        : WORD;
   wDayOfWeek    : WORD;
   wDay          : WORD;
   wHour         : WORD;
   wMinute       : WORD;
   wSecond       : WORD;
   wMilliseconds: WORD;
END_STRUCT
END_TYPE
```

wYear : Specifies the year: 1970 ~ 2106;

wMonth : Specifies the month: 1 ~ 12 (January = 1, February = 2 and so on);

wDayOfWeek : Specifies the day of the week: 0 ~ 6 (Sunday = 0, Monday = 1 and so on);

wDay : Specifies the day of the month: 1 ~ 31;

wHour : Specifies the hour: 0 ~ 23;

wMinute : Specifies the minute: 0 ~ 59;

wSecond : Specifies the second: 0 ~ 59;

wMilliseconds : Specifies the millisecond: 0 ~ 999;

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 and above	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.2 OTSTRUCT

```
TYPE OTSTRUCT
STRUCT
   wWeek         : WORD;
   wDay          : WORD;
   wHour         : WORD;
   wMinute       : WORD;
   wSecond       : WORD;
   wMilliseconds : WORD;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0 and above	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Development environment	Target system type	PLC libraries to include
		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)

5.3 PROFILERSTRUCT

```
TYPE PROFILERSTRUCT:
STRUCT
   LastExecTime     :DWORD;
   MinExecTime      :DWORD;
   MaxExecTime      :DWORD;
   AverageExecTime  :DWORD;
   MeasureCycle     :DWORD;
END_STRUCT
END_TYPE
```

LastExecTime: The most recently measured value for the execution time in [µs].

MinExecTime: The minimum execution time in [µs].

MaxExecTime: The maximum execution time in [µs].

AverageExecTime: The mean execution time for the last 10 measurements in [µs].

MeasureCycle: The number of measurements that have already been carried out.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.4 REMOTEPC

```
TYPE REMOTEPC
STRUCT
    NetId   : T_AmsNetId;
    Name    : STRING(31);
END_STRUCT
ND_TYPE
```

NetId: The network address of the remote PC;

Name: The remote PC identifier;

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.5 REMOTEPCINFOSTRUCT

```
TYPE REMOTEPCINFOSTRUCT
    ARRAY[0..99] OF REMOTEPC;
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Also see about this

REMOTEPC [▶ 231]

5.6 SYMINFOSTRUCT

```
TYPE SYMINFOSTRUCT
STRUCT
    symEntryLen         : UDINT;
    idxGroup            : UDINT;
    idxOffset           : UDINT;
    byteSize            : UDINT;
    adsDataType         : ADSDATATYPEID;
    symDataType         : T_MaxString;
    symComment          : T_MaxString;
END_STRUCT
END_TYPE
```

symEntryLen: The actual length in bytes of the symbol entry in the symbol table. The symbols are stored in a symbol table. The length of the individual entries is variable, and depends on the length of the symbol name, the type identifier, and the comment.

idxGroup: The index group of the symbolic variables;

idxOffset: The index offset of the symbolic variables;

byteSize: The amount of memory, in bytes, occupied by the value of the symbolic variables. A boolean PLC variable, for instance, occupies one byte, while a string with 20 characters in fact occupies 21 bytes (20 bytes for characters plus one byte for the zero marking the end of the string);

adsDataType: The ADS data type ID [▶ 233]. This type identifier is used in ADS access to symbolic variables. All PLC structures and arrays (user-defined data types) have the ADS data type identifier ADST_BIGTYPE, and can not be identified through this data type constant. In order to be able to identify the user-defined data types, use the *symDataType* variable, or read the base type of the individual variables in the structure.

symDataType: The data type identification of the symbolic variable as a string. For instance this might be the type name of a PLC data structure defined by the user (max. 255 characters).

symComment: A comment on the symbolic variable that the user has added to the PLC variable definition line (max. 255 characters).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.7 ADSDATATYPEID

```
TYPE ADSDATATYPEID:
(
    ADST_VOID           := 0,
    ADST_INT8           := 16,
    ADST_UINT8          := 17,
    ADST_INT16          := 2,
    ADST_UINT16         := 18,
    ADST_INT32          := 3,
    ADST_UINT32         := 19,
    ADST_INT64          := 20,
    ADST_UINT64         := 21,
    ADST_REAL32         := 4,
    ADST_REAL64         := 5,
    ADST_BIGTYPE        := 65,
    ADST_STRING         := 30,
    ADST_WSTRING        := 31,
    ADST_REAL80         := 32,
    ADST_BIT            := 33,
    ADST_MAXTYPES
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.8 E_RegValueType

```
TYPE E_RegValueType :
(
    REG_NONE                := 0,     (* No value TYPE *)
    REG_SZ,                  (* Unicode nul terminated STRING *)
    REG_EXPAND_SZ,              (* Unicode nul terminated STRING (with environment variable referenc
es) *)
    REG_BINARY,                 (* Free form binary *)
    REG_DWORD,                  (* 32-bit number and REG_DWORD_LITTLE_ENDIAN (same as REG_DWORD) *)
    REG_DWORD_BIG_ENDIAN,           (* 32-bit number *)
    REG_LINK,                   (* Symbolic Link (unicode) *)
    REG_MULTI_SZ,               (* Multiple Unicode strings *)
    REG_RESOURCE_LIST,              (* Resource list in the resource map *)
    REG_FULL_RESOURCE_DESCRIPTOR,       (* Resource list in the hardware description *)
    REG_RESOURCE_REQUIREMENTS_LIST,     (* *)
    REG_QWORD                       (* 64-
```

```
bit number and REG_QWORD_LITTLE_ENDIAN (same as REG_QWORD) *)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0, Build > 519	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0, Build >723	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.9 E_ArgType

```
TYPE E_ArgType :(

    ARGTYPE_UNKNOWN         := 0,
    ARGTYPE_BYTE,
    ARGTYPE_WORD,
    ARGTYPE_DWORD,
    ARGTYPE_REAL,
    ARGTYPE_LREAL,
    ARGTYPE_SINT,
    ARGTYPE_INT,
    ARGTYPE_DINT,
    ARGTYPE_USINT,
    ARGTYPE_UINT,
    ARGTYPE_UDINT,
    ARGTYPE_STRING,
    ARGTYPE_BOOL,
    ARGTYPE_BIGTYPE,
    ARGTYPE_BIGTYPE, (* byte buffer *)
    ARGTYPE_ULARGE, (* unsigned 64 bit ingeger (T_ULARGE_INTEGER), implemented in TwinCAT 2.10 Build
 > 1328 *)
    ARGTYPE_UHUGE, (* unsigned 128 bit integer (T_UHUGE_INTEGER), implemented in TwinCAT 2.10 Build
> 1328 *)
    ARGTYPE_LARGE, (* signed 64 bit integer (T_LARGE_INTEGER), implemented in TwinCAT 2.10 Build > 1
328 *)
    ARGTYPE_HUGE (* signed 128 bit integer (T_HUGE_INTEGER), implemented in TwinCAT 2.10 Build > 132
8 *)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.10 E_AmsLoggerMode

```
TYPE E_AmsLoggerMode :
(
    AMSLOGGER_RUN       := 1,   (*Start AMS logger*)
    AMSLOGGER_STOP      := 2    (*Stop AMS logger*)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747	PC or CX (x86)	TcUtilities.Lib

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >= 959		(Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.11 E_PersistentMode

Persistent data write mode. See for additional info: Writing of persistent data: System behaviour [▶ 277].

```
TYPE E_PersistentMode :
(
    SPDM_2PASS      := 0,
    SPDM_VAR_BOOST  := 1
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0, Build >= 959	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.12 E_TypeFieldParam

```
TYPE E_TypeFieldParam:(
    TYPEFIELD_UNKNOWN       := 0,
    TYPEFIELD_B,    (* b or B: binary number *)
    TYPEFIELD_O,    (* o or O: octal number *)
    TYPEFIELD_U,    (* u or U: unsigned decimal number *)
    TYPEFIELD_C,    (* c or C: one ASCII character *)
    TYPEFIELD_F,    (* f or F: float number ( normalized format )*)
    TYPEFIELD_D,    (* d or D: signed decimal number  *)
    TYPEFIELD_S,    (* s or S: string *)
    TYPEFIELD_XU,   (* X: hecadecimal number (upper case characters )*)
    TYPEFIELD_XL,   (* x: hecadecimal number (lower case characters )*)
    TYPEFIELD_EU,   (* E: float number ( scientific format ) *)
    TYPEFIELD_EL    (* e: float number ( scientific format ) *)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.13 E_EnumCmdType

Enumeration command parameter used by enumeration function blocks. Not all parameters are supported by every function block!

```
TYPE E_EnumCmdType :
(
    eEnumCmd_First := 0,
    eEnumCmd_Next,
    eEnumCmd_Abort
);
END_TYPE
```

eEnumCmd_First: Enumerate first element.

eEnumCmd_Next: Enumerate next element.

eEnumCmd_Abort: Command to abort enumeration (closes opened handles).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0, Build > 1033 TwinCAT v2.10.0, Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.14 E_RouteTransportType

```
TYPE E_RouteTransportType :
(
    eRouteTransport_None := 0,
    eRouteTransport_TCP_IP := 1,
    eRouteTransport_IIO_LIGHTBUS := 2,
    eRouteTransport_PROFIBUS_DP := 3,
    eRouteTransport_PCI_ISA_BUS := 4,
    eRouteTransport_ADS_UDP := 5,
    eRouteTransport_FATP_UDP := 6,
    eRouteTransport_COM_PORT := 7,
    eRouteTransport_USB := 8,
    eRouteTransport_CAN_OPEN := 9,
    eRouteTransport_DEVICE_NET := 10,
    eRouteTransport_SSB := 11,
    eRouteTransport_SOAP := 12
);
END_TYPE
```

The transport layer with which the AMS messages are carried. At present, only TCP/IP is supported as the transport layer.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0, Build > 1033 TwinCAT v2.10.0, Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.15 E_NumGroupTypes

Numeric group numbers.

```
TYPE E_EnumCmdType :
(
    eNumGroup_Float,
    eNumGroup_Unsigned,
    eNumGroup_Signed
);
END_TYPE
```

eNumGroup_Float: Floating point numbers.

eNumGroup_Unsigned: Unsigned numbers.

eNumGroup_Signed: Signed numbers.

Requirements

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

5.16 E_MIB_IF_Type

Management Information Base interface type.

```
TYPE E_MIB_IF_Type:
(
    MIB_IF_TYPE_OTHER := 1,
    MIB_IF_TYPE_ETHERNET := 6,
    MIB_IF_TYPE_TOKENRING := 9,
    MIB_IF_TYPE_FDDI := 15,
    MIB_IF_TYPE_PPP := 23,
    MIB_IF_TYPE_LOOPBACK := 24,
    MIB_IF_TYPE_SLIP := 28
);
END_TYPE
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.17 E_TimeZoneID

```
TYPE E_TimeZoneID:
(
    eTimeZoneID_Invalid := -1,
    eTimeZoneID_Unknown := 0,
    eTimeZoneID_Standard := 1,
    eTimeZoneID_Daylight := 2
);
END_TYPE
```

Additional information about operating system time zone settings.

- eTimeZoneID_Invalid = The time zone configuration could not be read;
- eTimeZoneID_Unknown = The system cannot determine the current time zone. This value is returned if daylight savings time is not used in the current time zone, because there are no transition dates.
- eTimeZoneID_Standard = The system is operating in the range covered by the *StandardDate* member of the **ST_TimeZoneInformation** structure.
- eTimeZoneID_Daylight = The system is operating in the range covered by the *DaylightDate* member of the **ST_TimeZoneInformation** structure.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1319	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.18 E_SBCSType

Windows SBCS (Single Byte Character Set) code page types.

```
TYPE E_SBCSType :
(
    eSBCS_WesternEuropean := 1, (* Windows 1252 (default) *)
    eSBCS_CentralEuropean := 2 (* Windows 1251 *)
);
END_TYPE
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

5.19 E_DbgContext

This variable type defines the context of the debug output and is used by protocol function blocks.

```
TYPE E_DbgContext:
(
    eDbgContext_NONE := 0,(* Not used *)
    eDbgContext_USER := 1,(* Service user *)
    eDbgContext_PROV := 2 (* Service provider *)
);
END_TYPE
```

Value	Description
eDbgContext_NONE	Parameter is not used
eDbgContext_USER	Debug output is triggered by service user
eDbgContext_PROV	Debug output is triggered by service provider

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build > 1537	PC or CX (x86, ARM)	TcUtilities.Lib

5.20 E_DbgDirection

This variable type can be used by buffer blocks or protocol blocks for configuration of debug outputs.

The debug output itself can be created e.g. by ADSLOGSTR function.

At a ringbuffer e.g. the debug output via the variable can be done in the following way:

- At the value *eDbgDirection_IN* or *eDbgDirection_ALL* the debug outbut is done if a new value is added into the buffer;
- At the value *eDbgDirection_OUT* or *eDbgDirection_ALL* the debug output is done if a value is deleted from the buffer;

```
TYPE E_DbgDirection:
(
    eDbgDirection_OFF       := 0,(* Disabled (no debug oputput) *)
    eDbgDirection_IN    := 1,(* Enabled only for incomming data *)
    eDbgDirection_OUT       := 2,(* Enabled only for outgoing data *)
    eDbgDirection_ALL       := 3(* Enabled for incomming and outgoing data *)
);
END_TYPE
```

Value	Description
eDbgDirection_OFF	Debug output is deactivated
eDbgDirection_IN	Activates the debug output for incomming data
eDbgDirection_OUT	Activates the debug output for outgoing data
eDbgDirection_ALL	Activates the debug output for incomming and outgoing data

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build > 1537	PC or CX (x86, ARM)	TcUtilities.Lib

5.21 T_Arg

```
TYPE T_Arg :
STRUCT
    eType   : E_ArgType     := ARGTYPE_UNKNOWN;     (* Argument data type *)
    cbLen   : UDINT         := 0;           (* Argument data byte length *)
    pData   : UDINT         := 0;           (* Pointer to argument data *)
END_STRUCT
END_TYPE
```

eType : Argument data type [▶ 234];

cbLen : Argument data byte length;

pData : Pointer to argument data;

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.22 T_FILETIME

```
TYPE T_FILETIME :
STRUCT
    dwLowDateTime   : DWORD;
    dwHighDateTime  : DWORD;
END_STRUCT
END_TYPE
```

The T_FILETIME structure is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.23 T_ULARGE_INTEGER

```
TYPE T_ULARGE_INTEGER :
STRUCT
    dwLowPart  : DWORD;
    dwHighPart : DWORD;
END_STRUCT
END_TYPE
```

Variable of this type represents unsigned 64 bit integer.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1030 TwinCAT v2.10.0 Build > 1231	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.24 T_UHUGE_INTEGER

```
TYPE T_UHUGE_INTEGER :
STRUCT
    qwLowPart  : T_ULARGE_INTEGER;(* Lower quad word *)
    qwHighPart : T_ULARGE_INTEGER;(* Higher quad word *)
END_STRUCT
END_TYPE
```

Variable of this type represents unsigned 128 bit integer.

Requirements

Development Environment	Target System Type	PLC Libraries to include
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_ULARGE_INTEGER [▶ 239]

5.25 T_LARGE_INTEGER

```
TYPE T_LARGE_INTEGER :
STRUCT
    dwLowPart  : DWORD;
    dwHighPart : DWORD;
END_STRUCT
END_TYPE
```

Variable of this type represents signed 64 bit integer.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

5.26 T_HUGE_INTEGER

```
TYPE T_HUGE_INTEGER :
STRUCT
    qwLowPart  : T_ULARGE_INTEGER;(* Lower quad word *)
    qwHighPart : T_ULARGE_INTEGER;(* Higher quad word *)
END_STRUCT
END_TYPE
```

Variable of this type represents signed 128 bit integer.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Also see about this

- T_ULARGE_INTEGER [▶ 239]

5.27 T_FIX16

```
TYPE T_FIX16 :
STRUCT
    value   : INT := 0; (* Internal fixed point number value. *)
    n       : WORD(0..15); (* Number of fractional bits. *)
    status  : DWORD := 0; (* Fixed point number status flags (reserved). *)
END_STRUCT
END_TYPE
```

value: This member variable contains the actual value of the fixed-point number (16 bits before and after the decimal point).

n: Number of decimal places. Permitted range: 0..15. The highest value bit is reserved for the sign bit.

status: Status flags (reserved, not used at the moment).

Variables of this type represent a signed 16 bit fixed-point number. This data type is often used by systems that have no FPU unit (e.g.: micro controllers or devices from the area of telecontrol). If for example data in fixed point number format has to be transferred via serial interface, then these data have to be converted to a suitable format.

The number of decimal places is chosen as per the required numerical range and resolution. For 15 decimal places it is possible to display fixed-point numbers in the range: -1..1-2^15. This corresponds approximately to the floating-point number range: -1..0.999969482421875.

Unlike the floating-point numbers the resolution of the fixed-point numbers is constant over the complete numerical range. Unfortunately the fixed-point numbers have a smaller number range for display. Care is to be taken with mathematical operations that can generate a positive or negative overflow.

Example 1:

An A/D-C supplies values as signed 16 bit fixed-point numbers with 15 decimal places. These measurement values are imported into the PLC and should be converted to LREAL data type.

```
PROGRAM FIX_TO_FLOAT
VAR
    adc_0 : WORD := 2#1010000000000000;(* = -0.75 (Q0.15) *)
    adc_1 : WORD := 2#0111000000000000;(* = +0.875 (Q0.15) *)

    fix_0, fix_1 : T_FIX16;
    dbl_0, dbl_1 : LREAL;
END_VAR
```

```
fix_0 := WORD_TO_FIX16( adc_0, 15 );
fix_1 := WORD_TO_FIX16( adc_1, 15 );

dbl_0 := FIX16_TO_LREAL( fix_0 );
dbl_1 := FIX16_TO_LREAL( fix_1 );
```

Example 2:

The parameters of a micro controller are signed 16 bit fixed-point numbers with 8 decimal places. The LREAL parameters in the PLC should be converted to this format.

```
PROGRAM FLOAT_TO_FIX
VAR
    dbl_0 : LREAL := +3.5;
    dbl_1 : LREAL := -3.5;

    fix_0, fix_1 : T_FIX16;
    ctrl_0, ctrl_1 : WORD;
END_VAR
```

```
fix_0 := LREAL_TO_FIX16( dbl_0, 8 );
fix_1 := LREAL_TO_FIX16( dbl_1, 8 );

ctrl_0 := FIX16_TO_WORD( fix_0 );
ctrl_1 := FIX16_TO_WORD( fix_1 );
```

Requirements

Development environment	Target system	PLC libraries to include
TwinCAT v2.10.0 Build >= 1326	PC or CX (X86, ARM)	TcUtilities.Lib

5.28 T_HHASHTABLE

A hash table handle.

```
TYPE T_HHASHTABLE :
STRUCT
    nCount  : UDINT := 0; (* number of used hash table entries *)
    nFree   : UDINT := 0; (* number of free hash table entries *)
END_STRUCT
END_TYPE
```

nCount:Number of used hash table elements.

nFree: Number of free hast table elements.

The hash table handle is used by the functionblock: FB_HashTableCtrl [▶ 190].

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

5.29 T_HLINKEDLIST

A Linked List Handle.

```
TYPE T_HLINKEDLIST :
STRUCT
    nCount  : UDINT := 0; (* number of used list nodes *)
    nFree   : UDINT := 0; (* number of free list nodes *)
END_STRUCT
END_TYPE
```

nCount: Number of elements occupied.

nFree: Number of free elements.

The Linked List Handle is used by the function block FB_LinkedListCtrl [▶ 192].

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcUtilities.Lib

5.30 T_HashTableEntry

A hash table entry/element.

```
TYPE T_HashTableEntry:
STRUCT
    key     : DWORD := 0;(* Entry key *)
    value   : DWORD := 0;(* Entry value *)
END_STRUCT
END_TYPE
```

key: (32 bit)

value: (32 bit).

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

5.31 T_LinkedListEntry

A linked list node/element.

```
TYPE T_LinkedListEntry:
STRUCT
    value   : DWORD := 0;(* Linked list value *)
END_STRUCT
END_TYPE
```

value: Value (32 bits; pointers are also possible).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcUtilities.Lib

5.32 ST_TcRouterStatusInfo

```
TYPE ST_TcRouterStatusInfo
STRUCT
    maxMem      : DWORD; (* Max. router memory byte size *)
    maxMemAvail : DWORD; (* Available router memory byte size *)
    regPorts    : DWORD; (* Number of registered ports *)
    regDrivers  : DWORD; (* Number of registered TwinCAT server ports *)
    amsDebugLog : BOOL; (* TRUE = Ams logging/debugging enabled, FALSE = Ams logging/
debugging disabled *)
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1235	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.33 ST_AmsRouteEntry

```
TYPE ST_TcRouterStatusInfo
STRUCT
    sName       : STRING(MAX_ROUTE_NAME_LEN);
    sNetID      : T_AmsNetId;
    sAddress    : STRING(MAX_ROUTE_ADDR_LEN);
    eTransport  : E_RouteTransportType;
    tTimeout    : TIME;
    dwFlags     : DWORD;
END_STRUCT
END_TYPE
```

sName: Symbolic name of the remote TwinCAT system. This name can be chosen freely. The maximum string length is limited through a constant (default: 31 characters).

sNetID: Network address of the remote TwinCAT system.

sAddress: System address in relation to the respective transport layer. If TCP/IP is used as the transport layer the IP address is specified here. The maximum string length is limited through a constant (default: 79 characters).

eTransport: Transport layer [▶ 236] used for carrying AMS messages. Currently only the TCP/IP transport layer is supported.

tTimeout: Timeout time. (reserved, not used).

dwFlags: Additional options (reserved, not used).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.34 ST_FindFileEntry

```
TYPE ST_FindFileEntry
STRUCT
    sFileName           : T_MaxString;
    sAlternateFileName  : STRING(13);
    fileAttributes      : ST_FileAttributes;
    fileSize            : T_ULARGE_INTEGER;
    creationTime        : T_FILETIME;
    lastAccessTime      : T_FILETIME;
    lastWriteTime       : T_FILETIME;
END_STRUCT
END_TYPE
```

sFileName: A null-terminated string that is the name of the file.

sAlternateFileName: A null-terminated string that is an alternative name for the file. This name is in the classic 8.3 (filename.ext) file name format.

fileAttributes: Specifies the file attributes [▶ 244] of the file or directory found.

fileSize: 64 bit unsigned integer [▶ 239]. The size of the file is equal to (dwHighPart * (0xFFFFFFFF+1)) + dwLowPart.

creationTime: For both files and directories, the structure [▶ 239] specifies when the file or directory was created.

lastAccessTime: For a file, the structure specifies when the file was last read from or written to.For a directory, the structure specifies when the directory was created.

lastWriteTime: For a file, the structure specifies when the file was last written to. For a directory, the structure specifies when the directory was created.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1302	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.35 ST_FileAttributes

File or directory attributes.

```
TYPE ST_FileAttributes
STRUCT
    bReadOnly          : BOOL;(* FILE_ATTRIBUTE_READONLY *)
    bHidden            : BOOL;(* FILE_ATTRIBUTE_HIDDEN *)
    bSystem            : BOOL;(* FILE_ATTRIBUTE_SYSTEM *)
    bDirectory         : BOOL;(* FILE_ATTRIBUTE_DIRECTORY *)
    bArchive           : BOOL;(* FILE_ATTRIBUTE_ARCHIVE *)
    bDevice            : BOOL;
(* FILE_ATTRIBUTE_DEVICE. Under CE: FILE_ATTRIBUTE_INROM or FILE_ATTRIBUTE_ENCRYPTED *)
    bNormal            : BOOL;(* FILE_ATTRIBUTE_NORMAL *)
    bTemporary         : BOOL;(* FILE_ATTRIBUTE_TEMPORARY *)
    bSparseFile        : BOOL;(* FILE_ATTRIBUTE_SPARSE_FILE *)
    bReparsePoint      : BOOL;(* FILE_ATTRIBUTE_REPARSE_POINT *)
    bCompressed        : BOOL;(* FILE_ATTRIBUTE_COMPRESSED *)
    bOffline           : BOOL;(* FILE_ATTRIBUTE_OFFLINE. Under CE: FILE_ATTRIBUTE_ROMSTATICREF *)
    bNotContentIndexed : BOOL; (* FILE_ATTRIBUTE_NOT_CONTENT_INDEXED. Under CE: FILE_ATTRIBUTE_ROMMO
DULE *)
    bEncrypted         : BOOL;(* FILE_ATTRIBUTE_ENCRYPTED *)
END_STRUCT
END_TYPE
```

bReadOnly: The file or directory is read-only. Applications can read the file but cannot write to it or delete it. In the case of a directory, applications cannot delete it.

bHidden: The file or directory is hidden. It is not included in an ordinary directory listing.

bSystem: The file or directory is part of the operating system or is used exclusively by the operating system.

bDirectory: The handle identifies a directory.

bArchive: The file or directory is an archive file or directory. Applications use this attribute to mark files for backup or removal.

bDevice: Reserved, not used.

bNormal: The file or directory has no other attributes set. This attribute is valid only if used alone.

bTemporary: The file is being used for temporary storage. File systems attempt to keep all of the data in memory for quicker access, rather than flushing it back to mass storage. A temporary file should be deleted by the application as soon as it is no longer needed.

bSparseFile: The file is a sparse file.

bReparsePoint: The file has an associated reparse point.

bCompressed: The file or directory is compressed. For a file, this means that all of the data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories.

bOffline: The file data is not immediately available. This attribute indicates that the file data has been physically moved to offline storage. This attribute is used by Remote Storage, the hierarchical storage management software.

bNotContentIndexed: The file is not be indexed by the content indexing service.

bEncrypted: The file or directory is encrypted. For a file, this means that all data in the file is encrypted. For a directory, this means that encryption is the default for newly created files and subdirectories.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1302	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.36 ST_IPAdapterInfo

Local adapter information..

```
TYPE ST_IPAdapterInfo
STRUCT
    bDefault        : BOOL;
    sAdapterName    : STRING(MAX_ADAPTER_NAME_LENGTH) := '';
    sDescription    : STRING(MAX_ADAPTER_DESCRIPTION_LENGTH) := '';
    physAddr        : ST_IPAdapterHwAddr;
    dwIndex         : DWORD;
    eType           : E_MIB_IF_Type;
    sIpAddr         : T_IPv4Addr;
    sSubNet         : T_IPv4Addr;
    sDefGateway     : T_IPv4Addr;
    bDhcpEnabled    : BOOL;
    sDhcpSrv        : T_IPv4Addr;
    bHaveWins       : BOOL;
    sPrimWinsSrv    : T_IPv4Addr;
    sSecWinsSrv     : T_IPv4Addr;
    tLeaseObt       : DT;
    tLeaseExp       : DT;
END_STRUCT
END_TYPE
```

bDefault: Supported only **under Windows CE!** Specifies whether this adapter is default TwinCAT adapter..

sAdapterName: Specifies the name of the adapter.

sDescription: Specifies a description for the adapter.

physAddr: Hardware address [▶ 246].

dwIndex: Specifies the internal adapter index.

eType: Specifies the adapter type [▶ 237].

sIpAddr: Specifies the IP address for this adapter.

sSubNet: Specifies the IP address mask.

sDefGateway: Specifies the IP address of the default gateway for this adapter.

bDhcpEnabled: Specifies whether dynamic host configuration protocol (DHCP) is enabled for this adapter.

sDhcpSrv: Specifies the IP address of the DHCP server for this adapter.

bHaveWins: Specifies whether this adapter uses Windows Internet Name Service (WINS).

sPrimWinsSrv: Specifies the IP address of the primary WINS server.

sSecWinsSrv: Specifies the IP address of the secondary WINS server.

tLeaseObt: Specifies the time when the current DHCP lease was obtained (UTC time).

tLeaseExp: Specifies the time when the current DHCP lease will expire. (UTC time).

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.37 ST_IPAdapterHwAddr

Physical address (MAC).

```
TYPE ST_IPAdapterHwAddr
STRUCT
    length  : UDINT := 0;
    b       : ARRAY[0..MAX_ADAPTER_ADDRESS_LENGTH] OF BYTE;
END_STRUCT
END_TYPE
```

length: Byte length of physical addres.

b: MAC address bytes.

Requirements

Development environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.38 ST_FileRBufferHead

FileBuffer-Header-Status. This structure is used by FB_FileRingBuffer [▶ 166]. The structure is read/stored every time the buffer is opened/closed. It is updated at each reading/writing of the data.

```
TYPE ST_FileRBufferHead
STRUCT
    status   : DWORD := 0; (* buffer status flags Bit 0 = 1 => Opened, Bit 0 = 0 => Closed, Bit 1 =
1 file corrupted, all other bits are reserved *)
    access   : UDINT := 0; (* access counter, increments every time the buffer is reopened *)
    nID      : UDINT := 0; (* user defined value *)
    cbBuffer : UDINT := 16#100000; (* max. buffer size (1MB) *)
    nCount   : UDINT := 0; (* number of fifo entries *)
    cbSize   : UDINT := 0; (* current (used) file buffer data byte length *)
    ptrFirst : UDINT := 0; (* seek pointer start position of first (oldest) buffer entry *)
    ptrLast  : UDINT := 0; (* seek pointer end position of last (newest) buffer entry *)
    rsrv0    : UDINT := 0; (* reserved *)
    rsrv1    : UDINT := 0; (* reserved *)
    rsrv2    : UDINT := 0; (* reserved *)
    rsrv3    : UDINT := 0; (* reserved *)
END_STRUCT
END_TYPE
```

status: Status flags. Bit 0 = 1 => file opened, Bit 0 = 0 => file closed. Bit 1 = 1 => file corrupted (file was closed incorrect, or max. buffer size does not match.).

acces: Access counter. Increments every time the buffer is reopened.

nID: User defined 32 bit value.

cbBuffer: Maximum buffer size.

nCount: Current number of stored fifo entries.

cbSize: Current number of stored data bytes.

ptrFirst: Seek pointer position of oldest buffer entry.

ptrLast: Seek pointer position of newest buffer entry.

rsrv0..rsrv3: reserved.

Requirements

Development Environment	Target System	PLC Libraires to include
TwinCAT v2.10.0 Build > 1313	PC or CX (x86, ARM)	TcUtilities.Lib

5.39 ST_TimeZoneInformation

Time zone information.

```
TYPE ST_TimeZoneInformation :
STRUCT
    bias         : DINT; (* Specifies the current bias, in minutes, for local time translation on co
mputer.
        The bias is the difference, in minutes, between Coordinated Universal Time (UTC) and local t
ime.
        UTC = local time + bias *)

    standardName : STRING(31); (* Specifies a string associated with standard time on computer. *)
```

```
    standardDate : TIMESTRUCT; (*Specifies a time structure that contains a date and local time
                when the transition from daylight saving time to standard time occurs on computer.*)

    standardBias : DINT; (* Specifies a bias value to be used during local time translations that oc
cur during standard time. *)

    daylightName : STRING(31); (* Specifies a string associated with daylight saving time on compute
r.
    For example, this member could contain "PDT" to indicate Pacific Daylight Time.*)

    daylightDate : TIMESTRUCT; (* Specifies a time structure that contains a date and local time whe
n the transition
                    from standard time to daylight saving time occurs on computer. *)

    daylightBias : DINT; (* Specifies a bias value to be used during local time translations that oc
cur during daylight saving time. *)
END_STRUCT
END_TYPE
```

The standard time is also called winter time. The bias parameter can also be negative values.

bias: Defines the current difference in minutes between local time and UTC time. UTC = local time + bias.

standardName: Description of the standard time as string.

standardDate: This structure [▶ 230] contains information about the transition from daylight-saving time (summer time) to standard time. The structure parameter *wMonth* is zero, if this value is unused. If this parameter is used, the *daylightDate*- parameter has to be used too. To be able to configure the *standardDate*- parameter set the *wYear*- parameter zero, select the desired day of week at *wDayOfWeek* and at *wDay* a value between 1 and 5 (week in the month, 5 is the last week in the month).

standardBias: Time difference in minutes for calculations of the local time during the standard time. This value is mostly zero.

daylightName: Description for the daylight-saving time (summer time) as string.

daylightDate: This structure contains information about the transition from standard time to daylight saving time (summer time). The structure parameter *wMonth* is zero, if this value is unused. If this parameter is used, the *standardDate*- parameter has to be used too. To be able to configure the *standardDate*- parameter set the *wYear*- parameter zero, select the desired day of week at *wDayOfWeek* and at *wDay* a value between 1 and 5 (week in the month, 5 is the last week in the month)

daylightBias: Time difference in minutes for calculation of the local time during the daylight-saving time.

Example:

FB_SetTimeZoneInformation [▶ 178].

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0 Build > 1319	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.40 GUID

System ID

```
TYPE GUID :
STRUCT
   Data1 : DWORD;
   Data2 : WORD;
   Data3 : WORD;
   Data4 : ARRAY[0..7] OFBYTE;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

6 Global variables/constants

6.1 Global Variables

VAR_GLOBAL

```
MAX_AVERAGE_MEASURES            :INT:=10; (*Possible values: 2..100*)

GLOBAL_FORMAT_HASH_PREFIX_TYPE  : E_HashPrefixTypes := HASHPREFIX_IEC; (* IEC prefixes for binary, o
ctal or hexadecimal type *)

GLOBAL_SBCS_TABLE : E_SBCSType  := eSBCS_WesternEuropean; (*Windows SBCS (Single Byte Character Set)
 Code Page Table *)

GLOBAL_DCF77_PULSE_SPLIT : TIME := T#140ms;(* 0 == pulse < 140ms, 1 == pulse > 140 *)

GLOBAL_DCF77_SEQUENCE_CHECK     : BOOL := FALSE;
(* TRUE = Enable plausibility check (two telegrams are checked), FALSE = Disable check *)

DEFAULT_CSV_FIELD_SEP           : BYTE := 16#3B;
(* semicolon (;) := 16#3B => german field separator, comma (,) := 16#2C => US field separator *)
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib; TcPlcUtilities.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

6.2 Format error codes

The following error codes are returned by the function block FB_FormatString [▶ 157] or the function F_FormatArgToStr [▶ 55]. If several arguments are used, the argument number (1..9) is returned in addition to the error code. The argument number provides information about where exactly an error was detected during formatting.

Error code	Meaning
16#00000000	No error
16#00000010 + Argument number (1..9)	Percent sign (%) at invalid position
16#00000020 + Argument number (1..9)	Asterisk parameter at invalid position
16#00000040 + Argument number (1..9)	Invalid width field value
16#00000080 + Argument number (1..9)	Invalid precision field value
16#00000100 + Argument number (1..9)	One of the flags at invalid position
16#00000200 + Argument number (1..9)	The width or precision field value at invalid position
16#00000400 + Argument number (1..9)	Dot "." sign of precision field at invalid position
16#00000800 + Argument number (1..9)	Invalid (unsupported) type field value
16#00001000 + Argument number (1..9)	Different type field and argument parameter
16#00002000 + Argument number (1..9)	Invalid format string parameters
16#00004000 + Argument number (1..9)	To much arguments in format string
16#00008000 + Argument number (1..9)	Destination string buffer overflow (formatted string is to long)

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

6.3 Error codes TcUtilities

The following error codes are returned by the function block FB_ScopeServerControl.

```
TYPE E_UTILITIES_ERRORCODES :
(
   eUtilError_NoError := 0,
   eUtilError_ScopeServerNotAvailable := 16#8001, (*
scope server not installed *)
   eUtilError_ScopeServerStateChange := 16#8002
);
END_TYPE
```

Requirements

Error code	Enum	Meaning
0x0000	eUtilError_NoError	No error
0x8001	eUtilError_ScopeServerNotAvailable	TwinCAT Scope Server is not available. Probably it is not installed
0x8002	eUtilError_ScopeServerStateChange	The requested state change in is not allowed. Refer FB_ScopeServerControl state diagram for the allowed transitions.

7 Samples

7.1 Example: Communication BC/BX<->PC/CX (F_SwapRealEx)

Here you can unpack the complete sources for PC or CX (x86) and BC (165): https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851042059.zip and https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851043467.zip

The example demonstrates the application of the F_SwapRealEx function. Load the PC (i386, pro file) application into the runtime system of a programming PC (i386) and the BC (165) application (165, pr6 file) into the runtime system of a bus controller (e.g. BC9000).

The PC application cyclically reads/writes a structure variable from/to the flag area of the BC. The structure variable contains REAL elements. Prior to application in the PC or transfer to the BC these elements have to be converted into the correct format. In order to be able to access the BC via ADS, it has to be entered as Remote PC in the list of TwinCAT AMS router connections (right mouse click on the TwinCAT system icon -> Properties -> AMS Router).

BC and PC have different data alignment. Define structures with 4-byte or better 8-byte alignment if they are to be used for the data exchange between BC and PC.

Structure variable definition on both systems:

```
TYPE ST_FrameData :
STRUCT
(*    8 byte aligned structure, byte size := 152 byte
    Define 8 Byte aligned structure to read/
write BC (Bus Terminal Controller) data from x86, x64, 165 or ARM CPU platform.

    TwinCAT 2 PC (x86) structures are 1 Byte - Aligned,
    TwinCAT 2 BC (165) structures are 2 Byte (WORD) - Aligned,
    TwinCAT 2 PC (ARM) structures are 4 Byte (DWORD) - Aligned,
    TwinCAT 3 PC (x86, x64, ARM) structures are 8 Byte - Aligned *)

    nFrameSize      : DWORD; (* Frame byte size, member byte size := 4 byte *)
    nTxFrames       : DWORD; (* Tx frame number, member byte size := 4 byte *)
    nRxFrames       : DWORD; (* Rx frame number, member byte size := 4 byte *)
    nCounter        : DWORD; (* Number value, member byte size := 4 byte *)
    fU          : REAL; (* Floating point number, member byte size := 4 byte *)
    fV          : REAL; (* Floating point number, member byte size := 4 byte *)
    fW          : REAL; (* Floating point number, member byte size := 4 byte *)
    aFloats         : ARRAY[0..9] OF REAL; (* Array of floating point numbers, array byte size := 40
 byte *)
 sMsg        : STRING; (* String variable, member byte size := 81 byte incl. string null delimiter *
)
 bEnable         : BOOL; (* Boolean flag, member byte size := 1 byte *)
 nRsv0       : BYTE; (* Reserved byte to meet the 8 byte alignment, member byte size := 1 byte *)
 nCRC        : BYTE; (* CRC checksum byte, member byte size := 1 byte *)
END_STRUCT
END_TYPE
```

PC or CX (x86) application:

Before the download the AMS-NetID of the BC must be configured in the program code. After a successful read operation the data length is checked, then a simple checksum. The REAL elements are then copied to the PC format. Rising edge at the bWrite variable activates the read command and rising edge at the bRead variable activates the write command.

```
PROGRAM MAIN
VAR
    bWrite      : BOOL;(* Rising edge writes data to the BC/BX (Bus Terminal Controller) *)
```

```
    bRead       : BOOL;(* Rising edge reads data from BC/BX (Bus Terminal Controller) *)
    stTxFrame           : ST_FrameData; (* Data transported from PC/CX (x86, ARM) to BC/
BX (Bus Terminal Contoroller) *)
    stRxFrame           : ST_FrameData; (* Data transported from BC/
BX (Bus Terminal Controller) to PC/CX (x86, ARM) *)
    fbWrite         : ADSWRITE :=  ( NETID := '172.17.61.50.1.1', PORT := 800, IDXGRP := 16#4020, ID
XOFFS := 500, TMOUT := DEFAULT_ADS_TIMEOUT );
    fbRead      : ADSREAD :=  ( NETID := '172.17.61.50.1.1', PORT := 800, IDXGRP := 16#4020, IDXOFFS
 := 0, TMOUT := DEFAULT_ADS_TIMEOUT );

  (* Temporary used variables *)
    stTxToBC        : ST_FrameData;
    stRxFromBC      : ST_FrameData;
    i               : INT;
    nTxState        : UDINT;
    nRxState        : UDINT;
    nTxErrors       : UDINT;
    nRxErrors       : UDINT;
END_VAR
```

```
(*#########################################################################################*)
CASE nTxState OF
    0:
        IF bWrite THEN(* Write BC/BX data *)
         bWrite := FALSE;
            (* Prepare/modify tx data *)
            stTxFrame.nFrameSize    := SIZEOF( stTxFrame );(* Set frame byte size *)
         stTxFrame.nTxFrames     := stTxFrame.nTxFrames + 1;(* Increment the send frame number *)
         stTxFrame.nRxFrames     := stRxFrame.nTxFrames;(* Report the received frame number *)
         stTxFrame.bEnable           := NOT stTxFrame.bEnable;(* Toggle bool flag *)
         stTxFrame.nCounter      := stTxFrame.nCounter + 1;(* Increment counter value *)
         stTxFrame.sMsg      := CONCAT( 'Message from PC/
CX, counter: ', DWORD_TO_STRING( stTxFrame.nCounter ) );(* Create some string message *)
         stTxFrame.fU        := stTxFrame.fU + 1.2;(* Modify some floating point values *)
         stTxFrame.fV        := stTxFrame.fV + 3.4;
            stTxFrame.fW        := stTxFrame.fW + 5.6;
            FOR i:= 0 TO 9 DO
                stTxFrame.aFloats[i] := stTxFrame.aFloats[i] + i;
            END_FOR
            stTxFrame.nCRC      := 0;

          (* Create temporary copy of tx data *)
         stTxToBC := stTxFrame;

          (* Swap REAL variables to BC/BX (Bus Terminal Controller) format *)
         F_SwapRealEx( stTxToBC .fU );
            F_SwapRealEx( stTxToBC .fV );
            F_SwapRealEx( stTxToBC .fW );
            FOR i:= 0 TO 9 DO
                F_SwapRealEx( stTxToBC .aFloats[i] );
            END_FOR

          (* Create CRC check number *)
         stTxToBC .nCRC  := F_CheckSum( ADR( stTxToBC ), SIZEOF( stTxToBC ) - 1 );

          (* Send *)
         fbWrite( WRITE := FALSE );
            fbWrite( LEN := SIZEOF( stTxToBC ), SRCADDR := ADR( stTxToBC ), WRITE := TRUE );
            nTxState := 1;
        END_IF

    1:(* Wait until ads write command not busy *)
     fbWrite( WRITE := FALSE );
        IF NOT fbWrite.BUSY THEN
            IF NOT fbWrite.ERR THEN
                nTxState := 0;
            ELSE(* Ads error *)
             nTxState := 100;
            END_IF
        END_IF

    100: (* TODO: Error state, add error handling *)
     nTxErrors := nTxErrors + 1;
        nTxState := 0;

END_CASE
```

```
(*########################################################################################*)
CASE nRxState OF
    0:
        IF bRead THEN(* Read BC/BX data *)
         bRead := FALSE;
            fbRead( READ := FALSE );
            fbRead( LEN := SIZEOF( stRxFromBC ), DESTADDR := ADR( stRxFromBC ), READ := TRUE );
            nRxState := 1;
        END_IF

    1:(* Wait until ads read command not busy *)
     fbRead( READ := FALSE );
        IF NOT fbRead.BUSY THEN
            IF NOT fbRead.ERR THEN
              (* Perform simple frame length check *)
             IF stRxFromBC.nFrameSize = SIZEOF( stRxFromBC ) THEN (* Check frame length *)
(* Perform simple CRC check *)
                 IF stRxFromBC.nCRC = F_CheckSum( ADR( stRxFromBC ), SIZEOF( stRxFromBC ) - 1 ) THEN

                      (* Swap REAL variables to PC/CX (x86) format *)
                     F_SwapRealEx( stRxFromBC.fU );
                        F_SwapRealEx( stRxFromBC.fV );
                        F_SwapRealEx( stRxFromBC.fW );
                        FOR i:= 0 TO 9 DO
                            F_SwapRealEx( stRxFromBC.aFloats[i] );
                        END_FOR

                        stRxFrame := stRxFromBC;
                        nRxState    := 0;

                    ELSE(* => Checksum error *)
                     nRxState := 100;
                    END_IF
                ELSE(* => Invalid frame length *)
                 nRxState := 100;
                END_IF
            ELSE(* => Ads error *)
             nRxState := 100;
            END_IF
        END_IF

    100: (* TODO: Error state, add error handling *)
     nRxErrors := nRxErrors + 1;
        nRxState := 0;

END_CASE
```

BC (165) application:

The checksum is checked after each write access from the PC. Random values are then generated and assigned a simple checksum.

```
PROGRAM MAIN
VAR
    stRxFrame AT%MB500     : ST_FrameData;(* Data transported from PC/CX (x86, ARM) to BC/
BX (Bus Terminal Controller) *)
 stTxFrame AT%MB0       : ST_FrameData;(* Data transported from BC/
BX (Bus Terminal Controller) to PC/CX (x86, ARM) *)
 nReceivedFrame      : UDINT;
    i               : INT;
    nRxErrors           : UDINT;
END_VAR
```

```
(* New frame from PC/CX received? *)
IF stRxFrame.nTxFrames <> nReceivedFrame THEN
    (* Frame length OK? *)
 IF stRxFrame.nFrameSize = SIZEOF( stRxFrame) THEN
        (* Checksum OK? *)
     IF stRxFrame.nCRC = F_CheckSum( ADR( stRxFrame), SIZEOF( stRxFrame) - 1 ) THEN (* => OK *)
(* Create/modify the tx data *)
         stTxFrame.nFrameSize    := SIZEOF( stTxFrame);(* Set frame byte size *)
         stTxFrame.nTxFrames     := stTxFrame.nTxFrames + 1;(* Increment the send frame number *)
         stTxFrame.nRxFrames     := stRxFrame.nTxFrames;(* Report the received frame number *)
         stTxFrame.bEnable           := NOT stRxFrame.bEnable;(* Toggle bool flag *)
         stTxFrame.nCounter      := stTxFrame.nCounter + 1;(* Send some counter value *)
         stTxFrame.sMsg      := CONCAT( 'Message from BC/
BX, counter:', DWORD_TO_STRING( stTxFrame.nCounter ) );(* Create any string message *)
         stTxFrame.fU        := stRxFrame.fU + 10.0;(* Modify some floating point values *)
```

```
         stTxFrame.fV        := stRxFrame.fV + 100.0;
            stTxFrame.fW        := stRxFrame.fW + 1000.0;
            FOR i:= 0 TO 9 DO
                stTxFrame.aFloats[i] := stTxFrame.aFloats[i] + i + 3.141592;
            END_FOR
            stTxFrame.nCRC      := F_CheckSum( ADR( stTxFrame), SIZEOF( stTxFrame) - 1 );
(* Create checksum *)
     ELSE(* => Checksum error *)
         nRxErrors := nRxErrors + 1;
        END_IF
    ELSE(* => Invalid frame length *)
     nRxErrors := nRxErrors + 1;
    END_IF
    nReceivedFrame := stRxFrame.nTxFrames;
END_IF
```

7.2 Example: File search (FB_EnumFindFileEntry, FB_EnumFindFileList)

Here you can unpack the complete sources: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851044875.zip

Example FB_EnumFindFileEntry:

In the local TwinCAT system all files should be listed in the following directory: C:\Windows\system32\. The file names should be written as messages into the TwinCAT System Manager logger output. It should be possible to cancel this process.

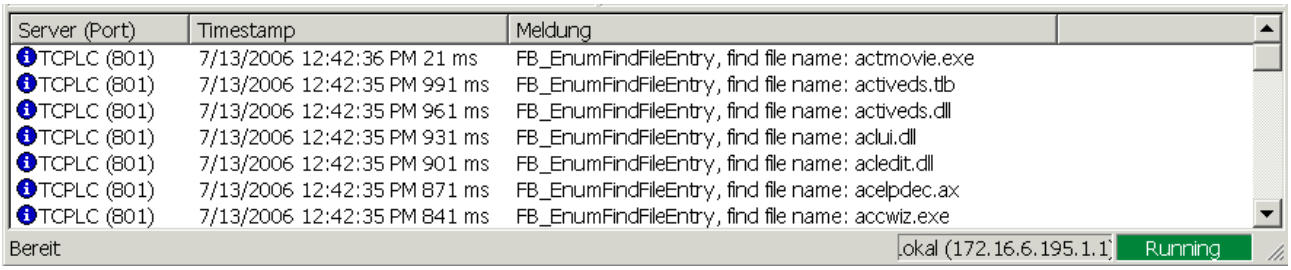
```
PROGRAM P_TestEnumEntry
VAR
    fbEnum : FB_EnumFindFileEntry := ( sNetID := '', tTimeout := T#5s, sPathName := 'C:
\Windows\System32\*.*' );
    bEnum : BOOL;
    bAbort : BOOL;
    nState : BYTE;
END_VAR
```

The listing of the found files begins with a rising edge at the *bEnum* variable. The process is canceled with a rising edge at the *bAbort* variable.

```
CASE nState OF
0:
    IF bEnum THEN (* flag set ? *)
     bEnum := FALSE; (* reset flag *)
        fbEnum.eCmd := eEnumCmd_First; (* enum first entry *)
        nState := 1;
    END_IF

1: (* enum one entry *)
 IF bAbort THEN
        bAbort := FALSE;
        fbEnum.eCmd := eEnumCmd_Abort;
    END_IF
    fbEnum( bExecute := FALSE );
    fbEnum( bExecute := TRUE );
    nState := 2;

2: (* wait until function block not busy *)
 fbEnum( bExecute := FALSE );
    IF NOT fbEnum.bBusy THEN
        IF NOT fbEnum.bError THEN
            IF NOT fbEnum.bEOE THEN
                ADSLOGSTR( ADSLOG_MSGTYPE_HINT OR ADSLOG_MSGTYPE_LOG, 'FB_EnumFindFileEntry, find fi
le name: %s', fbEnum.stFindFile.sFileName );
                fbEnum.eCmd := eEnumCmd_Next; (* enum next entry *)
             nState := 1;
            ELSE (* no more entries *)
             nState := 0;
            END_IF
        ELSE (* log error *)
         ADSLOGSTR( ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_LOG, 'FB_EnumFindFileEntry error:
%s', DWORD_TO_HEXSTR( fbEnum.nErrID, 0, FALSE ) );
            nState := 0;
```

```
        END_IF
    END_IF
END_CASE
```

Log messages in the TwinCAT System Manager logger output:



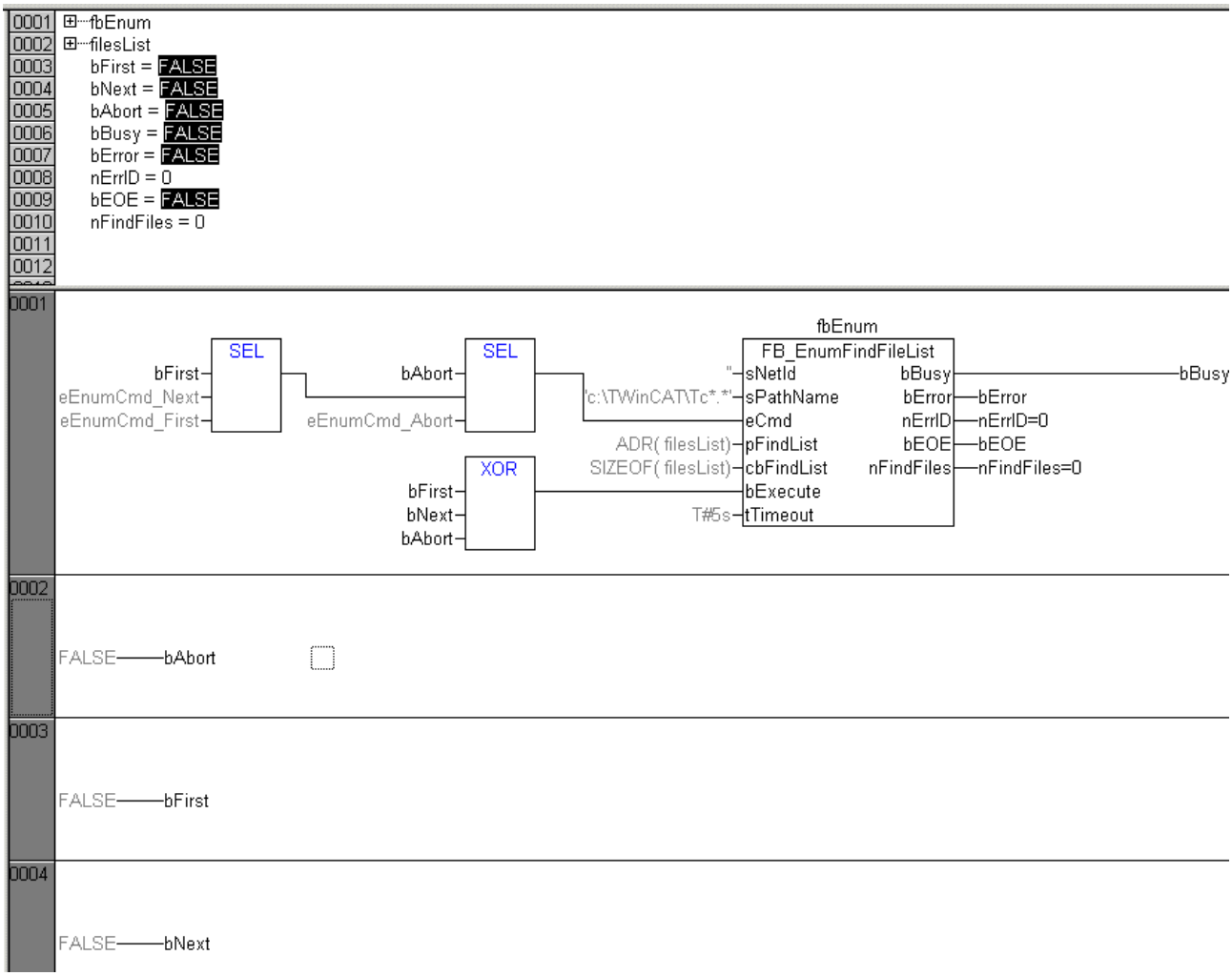
Server (Port)	Timestamp	Meldung
TCPLC (801)	7/13/2006 12:42:36 PM 21 ms	FB_EnumFindFileEntry, find file name: actmovie.exe
TCPLC (801)	7/13/2006 12:42:35 PM 991 ms	FB_EnumFindFileEntry, find file name: activeds.tlb
TCPLC (801)	7/13/2006 12:42:35 PM 961 ms	FB_EnumFindFileEntry, find file name: activeds.dll
TCPLC (801)	7/13/2006 12:42:35 PM 931 ms	FB_EnumFindFileEntry, find file name: aclui.dll
TCPLC (801)	7/13/2006 12:42:35 PM 901 ms	FB_EnumFindFileEntry, find file name: acledit.dll
TCPLC (801)	7/13/2006 12:42:35 PM 871 ms	FB_EnumFindFileEntry, find file name: acelpdec.ax
TCPLC (801)	7/13/2006 12:42:35 PM 841 ms	FB_EnumFindFileEntry, find file name: accwiz.exe

Bereit .okal (172.16.6.195.1.1) Running

Example FB_EnumFindFileList:

```
PROGRAM P_TestEnumList
VAR
    fbEnum      : FB_EnumFindFileList;
    filesList       : ARRAY[1..10] OF ST_FindFileEntry;
    bFirst      : BOOL;
    bNext       : BOOL;
    bAbort      : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    bEOE        : BOOL;
    nFindFiles      : UDINT;
END_VAR
```

Online view:



7.3 Example: File ring FiFo (FB_FileRingBuffer)

The complete source can be found here: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851046283.zip

The following example illustrates a simple application of the function block. The rising edge at *bOpen* opens an existing ring buffer file. If the file does not exist, a new one is created. The rising edge at *bClose* closes an open file. The rising edge at *bCreate* creates a new file. If you set *bAdd* = TRUE a new data record will be written into the ring buffer file, and when *bRemove* =TRUE the oldest data record is removed.

```
PROGRAM MAIN
VAR
    bOpen : BOOL;
    bClose : BOOL;
    bCreate : BOOL;
    bAdd : BOOL;
    bRemove : BOOL;
    bGet : BOOL;
    bReset : BOOL;

    fbFileBuffer : FB_FileRingBuffer := ( sNetId := '',
                    sPathName := 'c:\tmp\Data.dat',
                    ePath := PATH_GENERIC,
                    nID := 1,
                    cbBuffer := 100, (*cbBuffer := 16#80000000, 2GB*)
                    bOverwrite := TRUE,
                    pWriteBuff := 0,
```

```
                    cbWriteLen := 0,
                    pReadBuff := 0,
                    cbReadLen := 0,
                    tTimeout := t#5s );

    storeData : ARRAY[1..10] OF BYTE := 10(0);
    cbStore : UDINT;

    loadData : ARRAY[1..10] OF BYTE := 10(0);
    cbLoad : UDINT;

    i : INT;
END_VAR
```

```
fbFileBuffer( cbReturn => cbLoad );

IF NOT fbFileBuffer.bBusy THEN

    IF bOpen THEN
        bOpen := FALSE;
        fbFileBuffer.A_Open();
    END_IF

    IF bClose THEN
        bClose := FALSE;
        fbFileBuffer.A_Close();
    END_IF

    IF bCreate THEN
        bCreate := FALSE;
        fbFileBuffer.A_Create();
    END_IF

    IF bAdd THEN
        bAdd := FALSE;

        (* modify data *)
        FOR i:=1 TO 10 BY 1 DO
            storeData[i] := storeData[i] + 1;
        END_FOR

        cbStore := SEL( cbStore > 1, SIZEOF(storeData), cbStore -
 1 ); (* modify the data chunk length *)
        fbFileBuffer.A_AddTail( pWriteBuff := ADR(storeData), cbWriteLen := cbStore,
        pReadBuff := 0, cbReadLen:=0 );
    END_IF

    IF bRemove THEN
        bRemove := FALSE;
        fbFileBuffer.A_RemoveHead( pWriteBuff := 0, cbWriteLen := 0,
        pReadBuff := ADR(loadData), cbReadLen := SIZEOF(loadData));
    END_IF

    IF bGet THEN
        bGet := FALSE;
        fbFileBuffer.A_GetHead( pWriteBuff := 0, cbWriteLen := 0,
        pReadBuff := ADR(loadData), cbReadLen := SIZEOF(loadData));
    END_IF

    IF bReset THEN
        bReset := FALSE;
        fbFileBuffer.A_Reset();
    END_IF

END_IF
```

7.4 Example: Memory ring FiFo (FB_MemRingBuffer)

The complete source can be found here: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851047691.zip

The following example illustrates a simple application of the function block. Data records with the same length should be buffered (but not not mandatory). The data records have the following structure:

```
TYPE ST_DataSetEntry :
STRUCT
    bFlag : BOOL;
    nValue : BYTE;
    sMsg : STRING(20) := 'Unknown';
END_STRUCT
END_TYPE
```

Interface of function block FB_DataSetFifo:

The application specific function block used in the sample: FB_DataSetFifo uses internally the function block FB_MemRingBuffer. This function block simplifies the inserting/removing of data records. Furthermore the new function block supplies the current percental fill status of the buffer and an overwrite option.

If the intput *bOverwrite* is set and the buffer is full, the oldest entry will be removed from the buffer and overwritten by the newest one.

```
FUNCTION_BLOCK FB_DataSetFifo
VAR_INPUT
    bOverwrite : BOOL;
    in : ST_DataSetEntry;
END_VAR
VAR_OUTPUT
    bOk : BOOL;
    nCount : UDINT;
    nLoad : UDINT;
    out : ST_DataSetEntry;
END_VAR
VAR
    arrBuffer : ARRAY[0..MAX_BUFFER_SIZE] OF BYTE; (* Buffer memory used by FB_MemRingBuffer functio
n block *)
    fbBuffer : FB_MemRingBuffer;
END_VAR
VAR CONSTANT
    MAX_BUFFER_SIZE : UDINT := 1000;
END_VAR
```

The main program::

A rising edge at *bReset* deletes all buffer entries. If you set *bAdd* = TRUE a new data record will be written into the ring buffer file, and when *bRemove* =TRUE the oldest data record is removed. With a rising edge at *bGet* the oldest data record is read but not removed.

```
PROGRAM MAIN
VAR
    fbFifo : FB_DataSetFifo := ( bOverwrite := TRUE );
    newEntry : ST_DataSetEntry;
    oldEntry : ST_DataSetEntry;
    bSuccess : BOOL;
    nCount : UDINT;
    nLoad : UDINT;

    bReset : BOOL := TRUE;
    bAdd : BOOL := TRUE;
    bGet : BOOL := TRUE;
    bRemove : BOOL := TRUE;
END_VAR
```

```
IF bReset THEN
bReset := FALSE;
    (* reset fifo (clear all entries) *)
    fbFifo.A_Reset( in := newEntry, bOk=>bSuccess, nCount=> nCount, nLoad => nLoad );
END_IF

IF bAdd THEN
    bAdd := FALSE;

    (* create new or modify data set entry *)
    newEntry.bFlag := NOT newEntry.bFlag;
    newEntry.nValue := newEntry.nValue + 1;
    newEntry.sMsg := BYTE_TO_STRING(newEntry.nValue);

    (* add new entry to the fifo *)
    fbFifo.A_Add( in := newEntry, bOk=>bSuccess, nCount=> nCount, nLoad => nLoad );
END_IF
```

```
IF bGet THEN
    bGet := FALSE;
    (* get (but not delete) oldest entry *)
    fbFifo.A_Get( out => oldEntry, bOk => bSuccess, nCount => nCount, nLoad => nLoad );
END_IF

IF bRemove THEN
    bRemove:= FALSE;
    (* remove oldest entry *)
    fbFifo.A_Remove( out => oldEntry, bOk => bSuccess, nCount => nCount, nLoad => nLoad );
END_IF
```

7.5 Example: Memory ring FiFo (FB_MemRingBufferEx)

Here you can unpack the complete sources: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851049099.zip

A rising edge at bAdd causes new data elements (pubObj array) to be stored in the ring buffer. Via a rising edge at bGet the oldest data element can then be copied to the getObj variable.

Data elements that are not required are removed from the buffer via a rising edge at bRelease.

```
PROGRAM MAIN
VAR
    fbBuffer        : FB_MemRingBufferEx;
    buffer      : ARRAY[0..30] OF BYTE;
    bAdd, bGet, bRelease, bReset, bGetFree : BOOL;
    putObj      : ARRAY[0..3] OF BYTE := 16#00, 16#AA, 16#BB, 16#CC;
    getObj      : ARRAY[0..3] OF BYTE := 4(0);
    bOk         : BOOL;
    nCount      : UDINT;
    cbSize      : UDINT;
    cbFree      : UDINT;
END_VAR
```

```
IF bAdd THEN
    bAdd := FALSE;
    putObj[0] := putObj[0] + 1;(* modify data *)
    fbBuffer.A_AddTail(pBuffer := ADR( buffer ), cbBuffer := SIZEOF( buffer ),
            pWrite := ADR( putObj ), cbWrite := SIZEOF( putObj ),
            bOk=>bOk, nCount=>nCount, cbSize=>cbSize, cbFree=>cbFree );
    IF fbBuffer.bOk THEN
        ;(* Success *)
    ELSE
        ;(* Buffer overflow *)
    END_IF
END_IF

IF bGet THEN
    bGet := FALSE;
    fbBuffer.A_GetHead(pBuffer := ADR( buffer ), cbBuffer := SIZEOF( buffer ),
            bOk=>bOk, nCount=>nCount, cbSize=>cbSize, cbFree=>cbFree );
    IF fbBuffer.bOk THEN
        (* Success *)
        MEMCPY( ADR( getObj ), fbBuffer.pRead, MIN( SIZEOF( getObj ), fbBuffer.cbRead ) );
    ELSE
        ;(* Buffer empty *)
    END_IF
END_IF

IF bRelease THEN
    bRelease := FALSE;
    fbBuffer.A_FreeHead( pBuffer := ADR( buffer ), cbBuffer := SIZEOF( buffer ),
            bOk=>bOk, nCount=>nCount, cbSize=>cbSize, cbFree=>cbFree );
    IF fbBuffer.bOk THEN
        ;(* Success *)
    ELSE
```

```
        ;(* Buffer empty *)
    END_IF
END_IF

IF bGetFree THEN
    bGetFree := FALSE;
    fbBuffer.A_GetFreeSize(pBuffer := ADR( buffer ), cbBuffer := SIZEOF( buffer ),
            bOk=>bOk, nCount=>nCount, cbSize=>cbSize, cbFree=>cbFree );
END_IF

IF bReset THEN
    bReset := FALSE;
    fbBuffer.A_Reset(       pBuffer := ADR( buffer ), cbBuffer := SIZEOF( buffer ),
            bOk=>bOk,nCount=>nCount, cbSize=>cbSize, cbFree=>cbFree );
END_IF
```

7.6 Example: Hash table (FB_HashTableCtrl)

Here you can unpack the complete sources: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851050507.zip

The example project has two parts to the program:

- P_TABLE_OF_UDINT is a simple example program that simply edits 32-bit values in the hash table.
- P_TABLE_OF_STRUCTDATA illustrates how other data types (e.g. structured data types) can be managed in the context of the hash table.

The maximum number of table elements cannot be changed at runtime, and is limited in the example project by MAX_DATA_ELEMENTS. If you need more nodes, then you must increase the size of the table array accordingly (i.e. increase the value of the constant).

```
VAR_GLOBAL CONSTANT
    MAX_DATA_ELEMENTS : UDINT := 100;(* Max. number of elements in the list *)
    MAX_NAME_LENGTH : UDINT := 30;(* Max. length of article name *)
END_VAR
```

PROGRAM P_TABLE_OF_UDINT

In the first PLC cycle the article number and article name are stored in the table. The article number serves as key and the array index of the article name as value. Via a rising edge at bLookup the article name can be found via the article number.

```
PROGRAM P_TABLE_OF_UDINT
VAR
    sInfo       : T_MaxString := '';
    bAdd        : BOOL := TRUE;
    bLookup     : BOOL := TRUE;
    bRemove     : BOOL := TRUE;
    bEnum       : BOOL := TRUE;
    bCount      : BOOL := TRUE;

    search      : UDINT := 11111;(* article number *)

    fbTable         : FB_HashTableCtrl;(* basic hash table control function block *)
 hTable      : T_HHASHTABLE;(* hash table handle *)
    table       : ARRAY[0..MAX_DATA_ELEMENTS] OF T_HashTableEntry;
(* Max. number of hash table entries. The value of hash table entry = 32 bit integer *)
    names       : ARRAY[0..MAX_DATA_ELEMENTS] OF STRING(MAX_NAME_LENGTH);
    bInit       : BOOL := TRUE;
END_VAR
```

```
IF bInit THEN
    bInit := FALSE;
    F_CreateHashTableHnd( ADR( table ), SIZEOF( table ), hTable );(* Intialize table handle *)
 fbTable.A_Reset( hTable := hTable );
END_IF

IF bAdd THEN
    bAdd := FALSE;
```

```
    (* Fill table. Article number is the key. Array index number is the value (article name) *)
    names[0] := 'Chair';
    fbTable.A_Add( key := 12345, putValue := 0(* array index*), hTable := hTable );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF

    names[1] := 'Table';
    fbTable.A_Add( key := 67890, putValue := 1, hTable := hTable );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF

    names[2] := 'Couch';
    fbTable.A_Add( key := 11111, putValue := 2, hTable := hTable );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF

    names[3] := 'TV set';
    fbTable.A_Add( key := 22222, putValue := 3, hTable := hTable );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF
END_IF

IF bLookup THEN(* search for the article name by article number *)
 bLookup := FALSE;
    sInfo := '';
    fbTable.A_Lookup( key := search, hTable := hTable );
    IF fbTable.bOk THEN
        sInfo := names[fbTable.getValue];
    ELSE
        ;(* Entry not found *)
    END_IF
END_IF

IF bRemove THEN
    bRemove := FALSE;
    sInfo := '';
    fbTable.A_Remove( key := articleNo, hTable := hTable );
    IF fbTable.bOk THEN
        sInfo := names[fbTable.getValue];
    ELSE
        ;(* Entry not found *)
    END_IF
END_IF

IF bEnum THEN(* enumerate table entries *)
    bEnum := FALSE;
    sInfo := '';

    fbTable.A_GetFirst( putPosPtr := 0, hTable := hTable );
    IF fbTable.bOk THEN
        sInfo := names[fbTable.getValue];

        REPEAT
            fbTable.A_GetNext( putPosPtr := fbTable.getPosPtr , hTable := hTable );
            IF fbTable.bOk THEN
                sInfo := names[fbTable.getValue];
            END_IF
        UNTIL NOT fbTable.bOk
        END_REPEAT

    END_IF
END_IF

IF bCount THEN(* count entries in the table *)
    bCount := FALSE;
    sInfo := UDINT_TO_STRING( hTable.nCount );
END_IF
```

PROGRAM P_TABLE_OF_STRUCTDATA

This section of the program illustrates how structured data sets can be manipulated in the table in place of simply 32-bit numbers. In this case, the 32-bit element value is only used as a reference pointer to the actual value of the element. The reference pointer is able to point to instances of structured variables or other data types. The functionality is encapsulated in a function block. This function block, *FB_SpecialHashTableCtrl*, can be thought of as a specialised version of the function block *FB_HashTableCtrl*. The *FB_HashTableCtrl* block is also used internally by the specialised FB.

The *DATAELEMENT_TO_STRING* function is only used to permit visual output of the value of the element.

A structured variable of type *ST_DataElement* is used as an example. The highlight: You can add further member variables to the data type declaration of ST_DataElement without having to make any changes to the program or to the FB_SpecialHashTableCtrl function block.

The type declaration for ST_DataElement:

```
TYPE ST_DataElement :
STRUCT
    number  : UDINT := 0;
    name    : STRING(MAX_NAME_LENGTH) := '';
    price   : REAL := 0.0;

    (* add additional member variables here *)
END_STRUCT
END_TYPE
```

How do the 32-bit element values become reference pointers to the instances of the ST_DataElement array?

The maximum size of the table is limited by the constant MAX_DATA_ELEMENTS. It follows that no more than MAX_DATA_ELEMENTS reference pointers can be stored in the table. The *FB_SpecialHashTableCtrl* function block has an internal ST_DataElement array variable with the same size as the T_HashTableEntry array variable. For the sake of simplicity, the array indices in the two arrays are identical!

Each T_HashTableEntry array element can only be inserted into the table once. The *FB_HashTableCtrl* function block therefore searches for a free/unused T_HashTableEntry array element, and inserts it into the table if successful. The index of the T_HashTableEntry being used can be determined through the action *A_GetIndexAtPosPtr*. In the next step, the 32-bit node value that has just been added is assigned the address of the same array element in the ST_DataElement array. In the example project, this is done by calling the action *A_Add* again.

nodes[index].value := ADR(dataPool[index])

The assignment is, for instance, carried out in the *FB_SpecialHashTableCtrl->A_Add* action:

```
fbTable.A_Add( hTable := hTable, key := key, putValue := 16#00000000, getPosPtr=>getPosPtr, bOk=>bOk
 );
IF fbTable.bOk THEN
     fbTable.A_GetIndexAtPosPtr( hTable := hTable, putPosPtr := getPosPtr, getValue =>indexOfElem, b
Ok=>bOk );
    IF fbTable.bOk THEN
         pRefPtr := ADR( dataPool[indexOfElem] );

        pRefPtr^ := putValue;

         fbTable.A_Add( hTable := hTable, key := key, putValue := pRefPtr, bOk=>bOk );
        IF fbTable.bOk THEN
            getValue := putValue;
        END_IF
    END_IF
END_IF
```

```
PROGRAM P_TABLE_OF_STRUCTDATA
VAR
    sInfo   : T_MaxString := '';
    bAdd    : BOOL := TRUE;
    bLookup     : BOOL := TRUE;
```

```
    bRemove     : BOOL := TRUE;
    bEnum   : BOOL := TRUE;
    bCount  : BOOL := TRUE;

    search  : UDINT := 11111;(* article number *)

    fbTable     : FB_SpecialHashTableCtrl;(* Specialized hash table control function block *)
    putValue    : ST_DataElement;
    getValue    : ST_DataElement;
    getPosPtr : POINTER TO T_HashTableEntry := 0;
    bInit   : BOOL := TRUE;
END_VAR
```

```
IF bInit THEN
    bInit := FALSE;
    fbTable.A_Reset();(* reset / initialize table *)
END_IF


IF bAdd THEN
    bAdd := FALSE;

    (* Fill table. Article number is the key and data structure is the value *)
    putValue.number := 12345;
    putValue.name := 'Chair';
    putValue.price := 44.98;
    fbTable.A_Add( key := 12345, putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF

    putValue.number := 67890;
    putValue.name := 'Table';
    putValue.price := 99.98;
    fbTable.A_Add( key := 67890, putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF

    putValue.number := 11111;
    putValue.name := 'Couch';
    putValue.price := 99.98;
    fbTable.A_Add( key := 11111, putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF

    putValue.number := 22222;
    putValue.name := 'TV set';
    putValue.price := 99.98;
    fbTable.A_Add( key := 22222, putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbTable.bOk THEN
        ;(* Table overflow *)
    END_IF
END_IF

IF bLookup THEN(* search for the article name by article number *)
    bLookup := FALSE;
    sInfo := '';
    fbTable.A_Lookup( key := search, getPosPtr=>getPosPtr, getValue=>getValue );
    IF fbTable.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );
    ELSE
        ;(* Entry not found *)
    END_IF
END_IF

IF bRemove THEN(* remove one entry from the table *)
    bRemove := FALSE;
    sInfo := '';
    fbTable.A_Remove( key := search, getPosPtr=>getPosPtr, getValue=>getValue );
    IF fbTable.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );
    ELSE
        ;(* Entry not found *)
    END_IF
END_IF

IF bEnum THEN(* enumerate table entries *)
    bEnum := FALSE;
```

```
    sInfo := '';

    fbTable.A_GetFirst( putPosPtr := 0, getPosPtr=>getPosPtr, getValue=>getValue );
    IF fbTable.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );

        REPEAT
            fbTable.A_GetNext( putPosPtr := fbTable.getPosPtr , getPosPtr=>getPosPtr, getValue=>getV
alue );
            IF fbTable.bOk THEN
                sInfo := DATAELEMENT_TO_STRING( getValue );
            END_IF
        UNTIL NOT fbTable.bOk
        END_REPEAT
    END_IF
END_IF

IF bCount THEN(* count entries in the table *)
    bCount := FALSE;
    fbTable.A_Count();
    IF fbTable.bOk THEN
        sInfo := UDINT_TO_STRING( fbTable.nCount );
    END_IF
END_IF
```

7.7 Example: Linked list (FB_LinkedListCtrl)

Here you can unpack the complete sources: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851051915.zip

The example project has two parts to the program:

- P_LIST_OF_UDINT is a simple example program that simply edits 32-bit values in the linked list.
- P_LIST_OF_STRUCTDATA illustrates how other data types (e.g. structured data types) can be managed in the context of the linked list.

The maximum number of node elements cannot be changed at runtime, and is limited in the example project by MAX_DATA_ELEMENTS. If you need more nodes, then you must increase the size of the node array accordingly (i.e. increase the value of the constant).

```
VAR_GLOBAL CONSTANT
    MAX_DATA_ELEMENTS : UDINT := 100;(* Max. number of elements in the list *)
    MAX_NAME_LENGTH : UDINT := 30;(* Max. length of article name *)
END_VAR
```

PROGRAM P_LIST_OF_UDINT

The handle of the linked list is initialised in the first PLC cycle. This handle is then passed as the VAR_IN_OUT variable to the FB_LinkedListCtrl function block when accessing the list. The linked list is manipulated through the action calls of the function block. This allows node elements to be added, removed, searched and so on. The desired action is executed in response to a rising edge of the associated boolean variable. When you run the program, all the operations are carried out once.

```
PROGRAM P_LIST_OF_UDINT
VAR
    sInfo       : T_MaxString := '';
    bAddTailValue    : BOOL := TRUE;
    bAddHeadValue    : BOOL := TRUE;
    bGetTail         : BOOL := TRUE;
    bGetHead         : BOOL := TRUE;
    bFind       : BOOL := TRUE;
    bRemoveHeadValue     : BOOL := TRUE;
    bRemoveTailValue     : BOOL := TRUE;
    bCount      : BOOL := TRUE;

    search      : UDINT := 12345;

    fbList      : FB_LinkedListCtrl;(* basic linked list control function block *)
    hList       : T_HLINKEDLIST;(* linked list handle *)
```

```
    nodes       : ARRAY[0..MAX_DATA_ELEMENTS] OF T_LinkedListEntry;
(* Max. number of linked list nodes. The value of list node = 32 bit integer *)
    putValue        : UDINT;
    getValue        : UDINT;
    getPosPtr       : POINTER TO T_LinkedListEntry := 0;
    bInit       : BOOL := TRUE;
END_VAR
```

```
IF bInit THEN
    bInit   := FALSE;
    F_CreateLinkedListHnd( ADR( nodes ), SIZEOF( nodes ), hList );
    fbList.A_Reset( hList := hList );
END_IF

IF bAddTailValue THEN(* add some nodes to the list *)
    bAddTailValue := FALSE;
    putValue := 22222;
    fbList.A_AddTailValue( hList := hList, putValue := putValue, getPosPtr=>getPosPtr, getValue=>get
Value );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue := 11111;
    fbList.A_AddTailValue( hList := hList, putValue := putValue, getPosPtr=>getPosPtr, getValue=>get
Value );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue := 12345;
    fbList.A_AddTailValue( hList := hList, putValue := putValue, getPosPtr=>getPosPtr, getValue=>get
Value );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue := 67890;
    fbList.A_AddTailValue( hList := hList, putValue := putValue, getPosPtr=>getPosPtr, getValue=>get
Value );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF
END_IF

IF bAddHeadValue THEN
    bAddHeadValue := FALSE;
    putValue := 33333;
    fbList.A_AddHeadValue( hList := hList, putValue := putValue, getPosPtr=>getPosPtr, getValue=>get
Value );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue := 44444;
    fbList.A_AddHeadValue( hList := hList, putValue := putValue, getPosPtr=>getPosPtr, getValue=>get
Value );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF
END_IF

IF bGetTail THEN(* enumerate all nodes in list (start at tail node) *)
    bGetTail := FALSE;
    sInfo := '';
    fbList.A_GetTail( hList := hList, getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := UDINT_TO_STRING( getValue );
        REPEAT
            fbList.A_GetPrev( hList := hList, putPosPtr := getPosPtr, getValue=>getValue, getPosPtr=
>getPosPtr );
            IF fbList.bOk THEN
                sInfo := UDINT_TO_STRING( getValue );
            ELSE
                EXIT;
            END_IF
        UNTIL NOT fbList.bOk
        END_REPEAT
    END_IF
END_IF
```

```
IF bGetHead THEN(* enumerate all nodes in list (start at head node) *)
    bGetHead := FALSE;
    sInfo := '';
    fbList.A_GetHead( hList := hList, getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := UDINT_TO_STRING( getValue );
        REPEAT
            fbList.A_GetNext( hList := hList, putPosPtr := getPosPtr, getValue=>getValue, getPosPtr=
>getPosPtr );
            IF fbList.bOk THEN
                sInfo := UDINT_TO_STRING( getValue );
            ELSE
                EXIT;
            END_IF
        UNTIL NOT fbList.bOk
        END_REPEAT
    END_IF
END_IF

IF bFind  THEN(* search for node in the list by node value*)
    bFind   := FALSE;
    getPosPtr := 0;(* start from first node element *)
    sInfo := '';
    REPEAT
        fbList.A_FindNext( hList := hList, putPosPtr := getPosPtr, putValue := search, getValue=>get
Value, getPosPtr=>getPosPtr );
        IF fbList.bOk THEN
            sInfo := UDINT_TO_STRING( getValue );
        ELSE
            EXIT;
        END_IF
    UNTIL NOT fbList.bOk
    END_REPEAT
END_IF

IF bRemoveTailValue THEN(* remove tail node from node list *)
    bRemoveTailValue := FALSE;
    sInfo := '';
    fbList.A_RemoveTailValue( hList := hList, getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := UDINT_TO_STRING( getValue );
    END_IF
END_IF

IF bRemoveHeadValue THEN(* remove head node from node list *)
    bRemoveHeadValue := FALSE;
    sInfo := '';
    fbList.A_RemoveHeadValue( hList := hList, getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := UDINT_TO_STRING( getValue );
    END_IF
END_IF

IF bCount THEN(* count nodes in list *)
    bCount := FALSE;
    sInfo := UDINT_TO_STRING( hList.nCount );
END_IF
```

PROGRAM P_LIST_OF_STRUCTDATA

This section of the program illustrates how structured data sets can be manipulated in the list in place of simply 32-bit numbers. In this case, the 32-bit node value is only used as a reference pointer to the actual value of the node. The reference pointer is able to point to instances of structured variables or other data types. The functionality is encapsulated in a function block. This function block, *FB_SpecialLinkedListCtrl*, can be thought of as a specialised version of the function block *FB_LinkedListCtrl*. The *FB_LinkedListCtrl* block is also used internally by the specialised FB.

The *DATAELEMENT_TO_STRING* function is only used to permit visual output of the value of the node.

A structured variable of type *ST_DataElement* is used as an example. The highlight: You can add further member variables to the data type declaration of ST_DataElement without having to make any changes to the program or to the FB_SpecialLinkedListCtrl function block.

The type declaration for ST_DataElement:

```
TYPE ST_DataElement :
STRUCT
    number  : UDINT := 0;
    name    : STRING(MAX_NAME_LENGTH) := '';
    price   : REAL := 0.0;

    (* add additional member variables here *)
END_STRUCT
END_TYPE
```

A simple search function is implemented. You can search for nodes having a particular *name*, *number* or *price*.

How do the 32-bit node values become reference pointers to the instances of the ST_DataElement array?

The maximum size of the list is limited by the constant MAX_DATA_ELEMENTS. It follows that no more than MAX_DATA_ELEMENTS reference pointers can be stored in the list. The *FB_SpecialLinkedListCtrl* function block has an internal ST_DataElement array variable with the same size as the T_LinkedListEntry array variable. For the sake of simplicity, the array indices in the two arrays are identical!

Each T_LinkedListEntry array element can only be inserted into the list once. The *FB_LinkedListCtrl* function block therefore searches for a free/unused T_LinkedListEntry array element, and inserts it into the list if successful. The index of the T_LinkedListEntry being used can be determined through the action *A_GetIndexAtPosPtr*. In the next step, the 32-bit node value that has just been added is assigned the address of the same array element in the ST_DataElement array. In the example project, this is done by calling the action *A_SetValueAtPosPtr*.

nodes[index].value := ADR(dataPool[index])

The assignment is, for instance, carried out in the *FB_SpecialLinkedListCtrl->A_AddHeadValue* action:

```
fbList.A_AddHeadValue( hList := hList, putValue := 16#00000000, getPosPtr=>getPosPtr, bOk=>bOk );
IF fbList.bOk THEN
    fbList.A_GetIndexAtPosPtr( hList := hList, putPosPtr := getPosPtr, getValue =>indexOfElem, bOk=>
bOk );
    IF fbList.bOk THEN
        pRefPtr      := ADR( dataPool[indexOfElem] );
        pRefPtr^ := putValue;
        fbList.A_SetValueAtPosPtr( hList := hList, putPosPtr := getPosPtr, putValue := pRefPtr, bOk=
>bOk );
        IF fbList.bOk THEN
            getValue := putValue;
        END_IF
    END_IF
END_IF
```

```
PROGRAM P_LIST_OF_STRUCTDATA
VAR
    sInfo       : T_MaxString := '';
    bAddTailValue   : BOOL := TRUE;
    bAddHeadValue   : BOOL := TRUE;
    bGetTail        : BOOL := TRUE;
    bGetHead        : BOOL := TRUE;
    bFind       : BOOL := TRUE;
    bRemoveHeadValue    : BOOL := TRUE;
    bRemoveTailValue    : BOOL := TRUE;
    bCount      : BOOL := TRUE;
    search      : ST_DataElement := ( name := 'Couch', price := 99.98, number := 12345 );
(* search value ( by name, by price or by number ) *)
    eSearch         : E_SEARCH_CRITERIA := eSEARCH_BY_NAME;
(* / eSEARCH_BY_PRICE / eSEARCH_BY_NUMBER *)
    fbList      : FB_SpecialLinkedListCtrl;(* Specialized linked list control function block *)
    putValue        : ST_DataElement;
    getValue        : ST_DataElement;
```

```
    getPosPtr       : POINTER TO T_LinkedListEntry := 0;
    bInit       : BOOL := TRUE;
END_VAR
```

```
IF bInit THEN
    bInit   := FALSE;
    fbList.A_Reset();(* reset / initialize list *)
END_IF

IF bAddTailValue THEN(* add some nodes to the list *)
    bAddTailValue := FALSE;
    putValue.number := 22222;
    putValue.name := 'TV set';
    putValue.price :=  99.98;
    fbList.A_AddTailValue( putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue.number := 11111;
    putValue.name := 'Couch';
    putValue.price :=  99.98;
    fbList.A_AddTailValue( putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue.number := 12345;
    putValue.name := 'Chair';
    putValue.price := 44.98;
    fbList.A_AddTailValue( putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue.number := 67890;
    putValue.name := 'Table';
    putValue.price := 99.98;
    fbList.A_AddTailValue( putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF
END_IF

IF bAddHeadValue THEN
    bAddHeadValue := FALSE;
    putValue.number := 33333;
    putValue.name := 'Couch';
    putValue.price := 199.98;
    fbList.A_AddHeadValue( putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF

    putValue.number := 44444;
    putValue.name := 'Couch';
    putValue.price := 299.98;
    fbList.A_AddHeadValue( putValue := putValue, getPosPtr=>getPosPtr, getValue=>getValue );
    IF NOT fbList.bOk THEN
        ;(* List overflow *)
    END_IF
END_IF

IF bGetTail THEN(* enumerate all nodes in list (start at tail node) *)
    bGetTail := FALSE;
    sInfo := '';
    fbList.A_GetTail( getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );
        REPEAT
            fbList.A_GetPrev( putPosPtr := getPosPtr, getValue=>getValue, getPosPtr=>getPosPtr );
            IF fbList.bOk THEN
                sInfo := DATAELEMENT_TO_STRING( getValue );
            ELSE
                EXIT;
            END_IF
        UNTIL NOT fbList.bOk
        END_REPEAT
    END_IF
END_IF
```

```
IF bGetHead THEN(* enumerate all nodes in list (start at head node) *)
    bGetHead := FALSE;
    sInfo := '';
    fbList.A_GetHead( getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );
        REPEAT
            fbList.A_GetNext( putPosPtr := getPosPtr, getValue=>getValue, getPosPtr=>getPosPtr );
            IF fbList.bOk THEN
                sInfo := DATAELEMENT_TO_STRING( getValue );
            ELSE
                EXIT;
            END_IF
        UNTIL NOT fbList.bOk
        END_REPEAT
    END_IF
END_IF

IF bFind  THEN(* search for node in the list by node value (name, price, number... )*)
    bFind   := FALSE;
    getPosPtr := 0;(* start from first node element *)
    sInfo := '';
    REPEAT
        fbList.A_Find( eSearch := eSearch, putPosPtr := getPosPtr, putValue := search, getValue=>get
Value, getPosPtr=>getPosPtr );
        IF fbList.bOk THEN
            sInfo := DATAELEMENT_TO_STRING( getValue );
        ELSE
            EXIT;
        END_IF
    UNTIL NOT fbList.bOk
    END_REPEAT
END_IF

IF bRemoveTailValue THEN(* remove tail node from node list *)
    bRemoveTailValue := FALSE;
    sInfo := '';
    fbList.A_RemoveTailValue( getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );
    END_IF
END_IF

IF bRemoveHeadValue THEN(* remove head node from node list *)
    bRemoveHeadValue := FALSE;
    sInfo := '';
    fbList.A_RemoveHeadValue( getValue=>getValue, getPosPtr=>getPosPtr );
    IF fbList.bOk THEN
        sInfo := DATAELEMENT_TO_STRING( getValue );
    END_IF
END_IF

IF bCount THEN(* count nodes in list *)
    bCount := FALSE;
    sInfo := '';
    fbList.A_Count( );
    IF fbList.bOk THEN
        sInfo := UDINT_TO_STRING( fbList.nCount );
    END_IF
END_IF
```

7.8 Example: Writing/reading of CSV file

Here you can unpack the complete sources relating to the project example: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851053323.zip

CSV stands for comma-separated values. The following documentation describes how CSV files can be written and read with the aid of auxiliary PLC CSV functions. CSV files, which are basically text files, can store simply structured data sets that can be used for data exchange between two systems. This format enables storage of tables or lists of different lengths. A table row corresponds to a data set (or row) in the CSV file. A table cell corresponds to a data field in the CSV file.

General information on the supported CSV format

- Files in CSV format should have the extension **.csv**.
- The CRLF character (CR = Carriage Return, LF= Line Feed) is used to separate the individual data sets (rows) (Windows operating systems), i.e. each data set must be followed by a CRLF.
- The CSV file must end with a CRLF character.
- Binary data must be enclosed in single quotation marks. If no single quotation marks are used the data field may only contain numbers and/or letters.
- Data fields containing special characters/control characters are enclosed in double quotation marks. If the data field contains a double quotation mark a second double quotation mark is added.
- A special character is used for separating data fields (columns). The standard separator for the individual data fields used by the auxiliary functions is a semicolon. In Germany and Europe a semicolon used as a data field separator, in the USA a comma tends to be used. The separator can be changed from semicolon to comma via the global PLC variable **DEFAULT_CSV_FIELD_SEP**.
- Each data set should have the same number of data fields (columns).

Basic configuration of a CSV file with n columns and n rows (the CRLF characters are usually not visible and are indicated in the diagram with the letters **CRLF**).

```
"Field1Record1";"Field2Record1";     ...     ;"Field(n)Record1"CRLF
"Field1Record2";"Field2Record2";    ...     ;"Field(n)Record2"CRLF

...

"Field1Record(n)";"Field2Record(n)";    ...     ;"Field(n)Record(n)"CRLF
```

Available function blocks and functions

- STRING_TO_CSVFIELD [▶ 71], ARG_TO_CSVFIELD [▶ 72]: converts PLC data in a data field to CSV format;
- CSVFIELD_TO_STRING [▶ 69], CSVFIELD_TO_ARG [▶ 70]: converts data field in CSV format to PLC data;
- FB_CSVMemBufferWriter [▶ 228]: generates data sets in a byte buffer from several data fields;
- FB_CSVMemBufferReader [▶ 227]: splits data sets in a byte buffer into individual data fields;

Write/read CSV file in text mode or binary mode

A CSV file can be read or written in text or binary mode with the aid of the PLC function blocks for file access. Depending on the selected mode there are differences with advantages and disadvantages.

In 99% of cases the CSV files can be read/written in text mode. Binary mode is only required in rare cases.

	Text mode	Binary mode
Function block for the file read access	**FB_FileGets** (special feature: During read access this block automatically removes the CR character at the end before the last data set. The character has to be restored/re-inserted to ensure that the FB_CSVMemBufferReader block can interpret such a data set)	**FB_FileRead**

	Text mode	Binary mode
Function block for file write access	**FB_FilePuts** (special feature: During write access this block automatically adds a CR character at the end before the last data set. However, the FB_CSVMemBufferWriter also generates the CR characters. In order to avoid duplication of the character in the CSV file it must be removed from the buffer before the write access)	**FB_FileWrite**
Programming effort	Smaller	Greater
Special characters, non-printable control characters in the data field	Not permitted	Permitted
Maximum data set length that can be written/read	Limited to 253 characters (data set + CRLF), i.e. the data set length must not exceed 253 characters.	The maximum data set length is theoretically unlimited.
A complete data set can be written with the function block for write access	Yes	Yes
A complete data set can be read with the function block for read access	Yes A data set in a pure text file ends with CRLF. In such a file CRLF indicates the end of a line. The FB_FileGets function block reads the data up to CRLF.	No
Binary data in a data field	Not permitted	Permitted
Auxiliary functions for conversion of PLC data into CSV format and vice versa	CSVFIELD_TO_STRING [▶ 69] STRING_TO_CSVFIELD [▶ 71]	CSVFIELD_TO_ARG [▶ 70] ARG_TO_CSVFIELD [▶ 72]
Supported PLC variable types that can be written/read directly	T_MaxString (STRING with 255 characters), other data types must first be converted to string and then read/written as a data field in string format.	Any data types can be written/read
Sample code	P_TextModeRead() P_TextModeWrite()	P_BinaryModeRead() P_BinaryModeWrite()

Example project

The project example actually contains 4 examples: 2 for write/read access in text mode (preferred) and 2 for write/read access in binary mode (rare):

P_TextModeRead();

P_TextModeWrite();

P_BinaryModeRead();

P_BinaryModeWrite();

CSV files generated with the project example:

Data fields without binary data: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851054731.csv

Data fields contain binary data: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851056139.csv (please note that this file requires special software for correct interpretation)

Basic program sequence for reading a CSV file in text mode:

1st step: Open the CSV file in text mode (FB_FileOpen). If successful go to step 2.

2nd step: Read a row with the function block FB_FileGets. Append a CR character (see notes in the table). If successful go to step 3 go, if not go to step 4 (the end of the file was reached or an error has occurred).

3rd step: Parse the read row with the function block FB_CSVMemBufferReader. The individual data fields are read. Then go to step 2 and read the next row. Repeat steps 2 and 3 until the end of the file is reached or an error occurs.

4nd step: Close the CSV file (FB_FileClose).

Basic program sequence for writing a CSV file in text mode.

1st step: Open the CSV file in text mode (FB_FileOpen). If successful go to step 2.

2nd step: Use the function block FB_CSVMemBufferWriter to generate a new data set. The individual data fields are written into a buffer. This buffer may be a larger string. Remove the CR character at the end of the data set and go to step 3.

3rd step: Write a row with the function block FB_FilePuts. Repeat steps 2 and 3 until all data sets have been written. If yes go to step 4.

4nd step: Close the file (FB_FileClose).

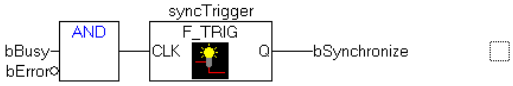
7.9 Example: Software clocks (RTC, RTC_EX, RTC_EX2)

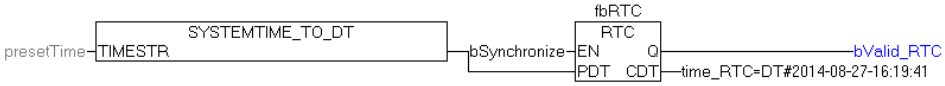
The complete sources can be found here: https://infosys.beckhoff.com/content/1033/tcplclibutilities/Resources/11851057547.zip

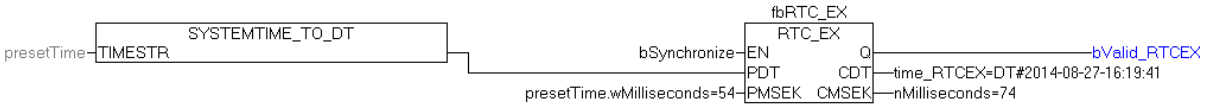
In the following example, the three software clocks are synchronized each 5 seconds with the local Windows system time (the local Windows system time is displayed in the task bar).

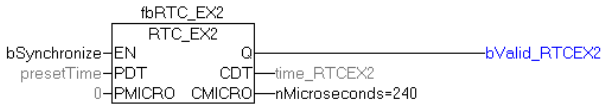
```
PROGRAM MAIN
VAR
    fbGetLocalTime  : NT_GetTime;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    presetTime      : TIMESTRUCT;

    syncTimer       : TON;
    syncTrigger     : F_TRIG;
    bSynchronize    : BOOL;

    fbRTC       : RTC;
    bValid_RTC      : BOOL;
    time_RTC    : DT;

    fbRTC_EX    : RTC_EX;
    bValid_RTCEX    : BOOL;
    time_RTCEX      : DT;
    nMilliseconds   : DWORD;

    fbRTC_EX2       : RTC_EX2;
    bValid_RTCEX2   : BOOL;
    time_RTCEX2     : TIMESTRUCT;
    nMicroseconds   : DWORD;
END_VAR
```

```
fbGetLocalTime
    bBusy = FALSE
    bError = FALSE
    nErrID = 0
presetTime
syncTimer
syncTrigger
    bSynchronize = FALSE
fbRTC
    bValid_RTC = TRUE
    time_RTC = DT#2014-08-27-16:19:41
fbRTC_EX
    bValid_RTCEX = TRUE
    time_RTCEX = DT#2014-08-27-16:19:41
    nMilliseconds = 74
fbRTC_EX2
    bValid_RTCEX2 = TRUE
time_RTCEX2
        wYear = 2014
        wMonth = 8
        wDayOfWeek = 3
        wDay = 27
        wHour = 16
        wMinute = 19
        wSecond = 41
        wMilliseconds = 74
    nMicroseconds = 240
```









8 Appendix

8.1 Format specification

This format specification is used by the function block FB_FormatString [▶ 157] and the function F_FormatArgToStr [▶ 55]. With the function block, the format specification is transferred via a string input variable. With the function, the individual function parameters are used.

The format specification containing the required and optional parameter fields has the following form:

%[ ] [ ] [.] Flags [▶ 276] Width [▶ 277] Precision [▶ 277] Type [▶ 275]

The simplest format specification only contains the percent sign and the type field (e.g. %s). All characters after the percent sign and up to the type field are interpreted as parameter fields. Characters to the left of the percent sign and after the type field are copied into the output string. Formatting is aborted with an error in the event of unidentifiable or illegal characters. Two consecutive percent signs (%%) can be used if the output string is to contain the percent sign.

Type

Required parameter field. This is an ASCII character that determines whether the associated argument is interpreted as a string, integer, or floating-point number. Please note that some type field parameters are case-sensitive.

Type	Argument	Output
b, B	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Binary string (e.g.: '101010111000').
o, O	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Octal string.
u, U	BYTE, WORD, DWORD, **SINT, **INT, **DINT, USINT, UINT, UDINT	Decimal string without sign.
c, C	BYTE, USINT	Single (ASCII) byte character.
f, F	***REAL, LREAL	Floating-point number. The string has the following form: **[-]dddd.dddd**, (dddd are decimal numbers). The number of digits before the decimal point depends on the value of the floating-point number. The number of digits after the decimal point depends on the required precision. The sign only appears for negative values. **'#INF'** is returned for infinite positive values, **'-#INF'** for infinite negative values. If the variable transferred has an illegal value (NaN, Not-a-Number), **'#QNAN'** or **'-#QNAN'** is returned. If the length of the formatted string exceeds the maximum permissible length of the resulting string, **'#OVF'** or **'-#OVF'** is returned.
d, D	BYTE, WORD, DWORD, SINT, INT, DINT, USINT, UINT, UDINT	Decimal string. The sign only appears for negative values.
s, S	STRING	Single byte character string. Characters are output up to the last zero or until the *precision* parameter is reached.
X	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Hexadecimal string. Upper case letters ('ABCDEF') are used for formatting.
x	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Hexadecimal string. Lower case letters ('abcdef') are used for formatting.

Type	Argument	Output
E	Not implemented. Reserved for future use!.	Floating-point numbers in scientific notation.
e	Not implemented. Reserved for future use!.	Floating-point numbers in scientific notation.

* The content of the REAL variable is returned as a binary, octal, hexadecimal or decimal string.

** The content of the signed types is returned as a binary, octal, hexadecimal or decimal string.

*** The REAL variable is converted to type LREAL and then formatted.

Flags

Optional parameter field. One or several flags can be specified in any order. These parameters determine the alignment of the formatted value and output of sign, blanks and the binary/octal/hex prefixes.

Flag	Meaning	Type	Standard
-	Left alignment flag. The formatted value is left aligned within the width aligned in parameter Width. **Width**.	Can be used in conjunction with all types.	Right alignment.
+	Sign flag. Forces output of the positive sign for signed positive numbers.	Only in conjunction with **e**, **E**, **f**, **F**, **d**, **D**. Otherwise the flag is ignored.	The negative sign only appears for negative values.
0	Zero flag. If this flag precedes the **Width** parameter, the resulting string is filled with zeros from the left until the required width is reached.	Only in conjunction with **e**, **E**, **f**, **F**, **s**, **S**. Otherwise the flag is ignored. The zero flag is also ignored if the left alignment flag (-) was set at the same time.	No filling with zeros.
Blank (' ')	Blank flag. A positive value is preceded by a blank	Only in conjunction with **e**, **E**, **f**, **F**, **d**, **D**. Otherwise the flag is ignored. The blank flag is also ignored if the left sign flag (**+**) was set at the same time	No blank.
#	Prefix flag. The formatted value is preceded by an "IEC" or a "standard C" prefix. "IEC" prefixes: **2#**, **8#**, **16# (default)** "Standard C" prefixes: **0**, **0x**, **0X**	Only in conjunction with **b**, **B**, **o**, **O**, **x**, **X**. Otherwise the flag is ignored. The "standard C" prefix type can be activated by setting the global variable GLOBAL_FORMAT_HASH_PREFIX_TYPE in the program: GLOBAL_FORMAT_HASH_PREFIX_TYPE := HASHPREFIX_STDC;	No prefix.

Width

Optional parameter field. This parameter must have a positive decimal value. It determines the minimum number of characters in the output string. Blanks (left or right, depending on the alignment flag) are added to the output string until the required width is reached. If the zero flag precedes the *Width* parameter, the resulting string is filled with zeros from the left until the required width is reached. However, the *Width* parameter never causes the output string to be cut to the desired length!

An asterisk (*) may also be used for the *Width* parameter. The required value is then provided by an argument (permissible types: BYTE, WORD, DWORD, USINT, UINT, UDINT). The argument for the *Width* parameter is then followed by the argument for the value to be formatted.

Precision

Optional parameter field. This parameter follows after the dot (.) and must have a positive decimal value. If the dot is not followed by a value, the default precision value is used (see table).

Type	Meaning	Standard
b, B, o, O, u, U, x, X, d, D	The precision parameter determines the number of digits in the output string. If there are not enough digits, the string is filled with zeros from the left. The output string is never cut.	Standard: 1
c, C	Has no meaning and is ignored.	Output: 1 character
f, F	The precision parameter determines the number of decimal places. The argument value is always rounded to the respective number of decimal places.	Standard: 6 decimal places
s, S	The precision parameter determines how many characters from the argument string are output. Characters exceeding the precision value are not output.	All characters up to the last zero are output.

An asterisk (*) may also be used for the *Precision* parameter. The required value is then provided by an argument (permissible types: BYTE, WORD, DWORD, USINT, UINT, UDINT). The argument for the *Precision* parameter is then followed by the argument for the value to be formatted.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

8.2 Writing of persistent data: System behaviour

Write trigger	Internal optimization of persistent data access	Persistent data consistency	Plc cycle time exceedance
Function block WritePersistentData [▶ 216]	None	All data is from same plc cycle.	Yes, if writing of all data takes more than plc cycle time.
Function block FB_WritePersistentData [▶ 152] and SPDM_2PASS [▶ 235]	Yes	All data is from same plc cycle.	Yes, if writing of all data takes more than plc cycle time.

Write trigger	Internal optimization of persistent data access	Persistent data consistency	Plc cycle time exceedance
Function block FB_WritePersistentData [▶ 152] and SPDM_VAR_BOOST [▶ 235]	Yes	The data of each variable is from same plc cycle.	Rare, if writing of biggest pers. variable takes more than plc cycle time.
TwinCAT system stop (all persistent data is written automatically on TwinCAT system stop).	Yes	All data is from same plc cycle.	None

Trademark statements

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Third-party trademark statements

Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

More Information:
www.beckhoff.com/tx1200

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