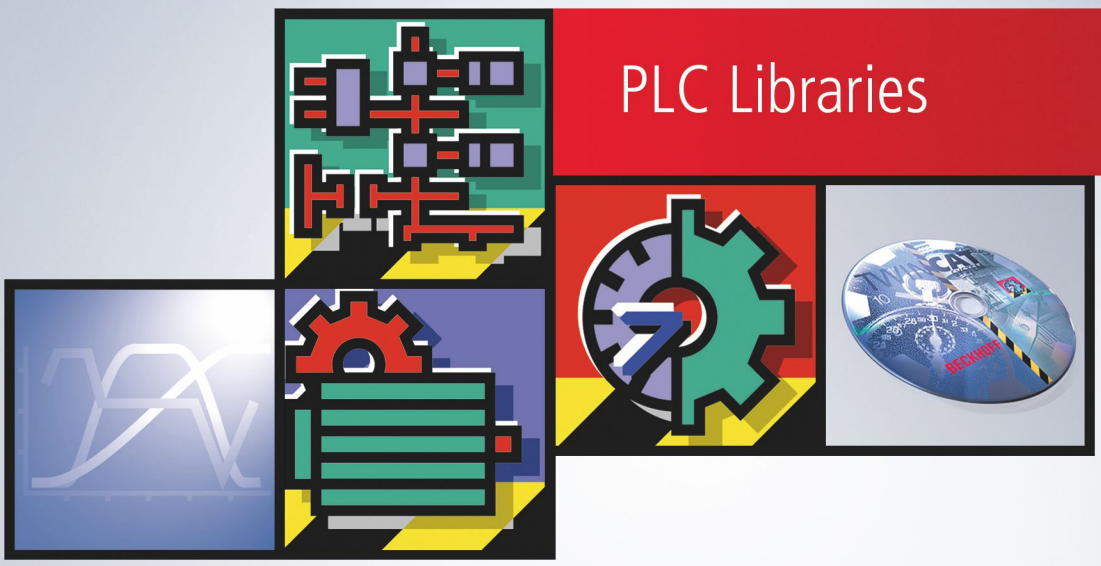
BECKHOFF New Automation Technology

Handbuch | DE

TX1200

TwinCAT 2 | PLC-Bibliothek: TcUtilities



15.11.2023 | Version: 1.3

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Die TcUtilities.Lib beinhaltet einige nützliche Funktionsbausteine und Funktionen, mit denen z.B. TwinCAT SPS- oder Betriebssystem-Funktionen aufgerufen werden können. Die meisten Funktionsbausteine basieren intern auf der System-Bibliothek.

Beim Hinzufügen der **TcUtilities.Lib** werden folgende Bibliotheken automatisch eingebunden: Standard.Lib; TcBase.Lib; TcSystem.Lib;

Betriebssystem Funktionen

Name	Beschreibung
NT_Shutdown [▶ 125]	Das Betriebssystem herunterfahren (shutdown)
NT_AbortShutdown [▶ 126]	Den Shutdown-Prozess unterbrechen
NT_Reboot [▶ 128]	Ein Restart des Betriebssystems durchführen
NT_GetTime [▶ 129]	Die aktuelle lokale Windows-Systemzeit lesen
NT_SetLocalTime [▶ 130]	Die aktuelle lokale Windows-Systemzeit setzen
NT_StartProcess [▶ 131]	Aus der SPS Windows-Anwendungen starten
NT_SetTimeToRTCTime [▶ 132]	Die lokale Windows-Systemzeit mit der Echtzeituhr des PCs synchronisieren
FB_RegQueryValue [▶ 141]	Werte aus der Registrierung auslesen
FB_RegSetValue [▶ 143]	Werte in die Registrierung schreiben
FB_EnumFindFileEntry [▶ 158]	Dieser Funktionsbaustein sucht in einem Verzeichnis nach einer Datei oder einem Unterverzeichnis dessen Name dem spezifizierten Namen gleicht. Die gefundenen Einträge können einzeln ausgelesen werden.
FB_EnumFindFileList [▶ 159]	Dieser Funktionsbaustein sucht in einem Verzeichnis nach einer Datei oder einem Unterverzeichnis dessen Name dem spezifizierten Namen gleicht. Die gefundenen Einträge können Gruppenweise ausgelesen werden.
FB_GetAdaptersInfo [▶ 165]	Liest Netzwerkadapterinformationen
FB_GetHostName [▶ 163]	Liest den Host-Namen des lokalen PCs
FB_GetHostAddrByName [▶ 163]	Konvertiert den Host-Namen in die (IPv4) Internet Protokoll Netzwerkadresse.
FB_GetTimeZoneInformation [▶ 177]	Liest die Zeitzone-Konfiguration des Betriebssystems
FB_SetTimeZoneInformation [▶ 179]	Setzt die Zeitzone-Konfiguration des Betriebssystems
FB_LocalSystemTime [▶ 180]	Liefert die lokale Windows-Systemzeit und Sommerzeit-/Winterzeit-Info

TwinCAT SPS Funktionen

Name	Beschreibung
PLC_Reset [▶ 133]	Reset der SPS durchführen
PLC_Start [▶ 134]	SPS starten
PLC_Stop [▶ 135]	SPS stoppen
PLC_ReadSymInfo [▶ 136]	Symbolinformationen der SPS lesen
PLC_ReadSymInfoByName [▶ 137]	Symbolinformationen einer SPS-Variablen anhand des Symbolnamens lesen
PLC_ReadSymInfoByNameEx [▶ 139]	Symbolinformationen einer SPS-Variablen anhand des Symbolnamens lesen. Der Kommentar wird abgeschnitten wenn die verfügbare Puffergröße nicht ausreichend ist.
Profiler [▶ 219]	Die Ausführungszeit vom SPS-Code ermitteln

Name	Beschreibung
WritePersistentData [▶ 218]	Schreiben der persistenten Daten auf den Datenträger aus der SPS auslösen
FB_WritePersistentData [▶ 151]	Schreiben der persistenten Daten auf den Datenträger aus der SPS auslösen (erweiterte Version)

Checksumme/CRC Funktionen

Name	Beschreibung
F_CheckSum16 [▶ 52]	Berechnet die 16 Bit Check Summe
F_DATA_TO_CRC16_CCITT [▶ 52]	Berechnet CRC16-CCITT (zyklische Redundanz Prüfung) von einem beliebigen Datentyp
F_BYTE_TO_CRC16_CCITT [▶ 53]	Berechnet CRC16-CCITT (zyklische Redundanz Prüfung) eines einzelnen Datenbytes

TwinCAT Systemfunktionen

Name	Beschreibung
TC_Restart [▶ 194]	TwinCAT System Restart durchführen
TC_Stop [▶ 195]	TwinCAT System stoppen
TC_Config [▶ 196]	TwinCAT System in den CONFIG-Modus versetzen
TC_CpuUsage [▶ 197]	Die CPU-Auslastung des TwinCAT Systems ermitteln
TC_SysLatency [▶ 198]	Die aktuelle und maximale Latenzzeit eines TwinCAT Systems ermitteln
GetRemotePCInfo [▶ 216]	Router Informationen über die konfigurierten Remote-PCs lesen
FB_GetLocalAmsNetId [▶ 153]	Die AmsNetId des lokalen TwinCAT PCs lesen
FB_GetRouterStatusInfo [▶ 145]	TwinCAT Router Statusinformationen lesen.
FB_EnumRouteEntry [▶ 146]	Router Verbindungsinformationen lesen
FB_AddRouteEntry [▶ 148]	Eine neue Router-Verbindung hinzufügen
FB_RemoveRouteEntry [▶ 149]	Eine vorhandene Router-Verbindung löschen

TwinCAT Scope View Funktionen

Mit den beschriebenen Funktionsbausteinen ist es möglich das TwinCAT Scope View über ADS-Kommandos zu bedienen. Da dafür ein fester ADS-Port erforderlich ist, kann nur die erste Instanz des TwinCAT Scope Views bedient werden. Einige der Funktionen können zur Zeit nur genutzt werden, wenn nur ein View in der Applikation aktiv ist.

Name	Beschreibung
ScopeLoadFile [▶ 199]	Scope View Configuration Project in den TwinCAT Scope View laden
ScopeSetOnline [▶ 200]	Scope View in Online-Zustand schalten
ScopeSetOffline [▶ 200]	Scope View in Offline-Zustand schalten
ScopeGetState [▶ 201]	Online/Offline - Zustand des Scope View ermitteln
ScopeManualTrigger [▶ 202]	Scope View manuell triggern
ScopeSetRecordLen [▶ 203]	Die Aufnahme-Zeit konfigurieren
ScopeGetRecordLen [▶ 203]	Die aktuelle Aufnahme-Zeit ermitteln
ScopeASCIIExport [▶ 204]	Scope View in ASCII-Datei exportieren
ScopeViewExport [▶ 205]	Scope View in binäre Datei exportieren
ScopeSaveAs [▶ 206]	Scope View Project unter einem bestimmten Dateinamen speichern
ScopeExit [▶ 207]	TwinCAT Scope View (exe) beenden

TwinCAT Scope Server

ab Bibliotheks-Version 2.0.52

Name	Beschreibung
FB_ScopeServerControl	Steuert (start/speichern..) den Scope Server für data logging

TwinCAT ADS Monitor Funktionen

Name	Beschreibung
FB_AmsLogger [▶ 154]	AMS Logger aus der SPS starten/stoppen.

Konvertierungsfunktionen

Name	Beschreibung
DT_TO_SYSTEMTIME [▶ 20]	DATE_AND_TIME in Windows Systemzeit-Struktur konvertieren
DT_TO_FILETIME [▶ 20]	DATE_AND_TIME in Windows File-Time konvertieren
SYSTEMTIME_TO_DT [▶ 21]	Windows Systemzeit-Struktur in DATE_AND_TIME konvertieren
SYSTEMTIME_TO_FILETIME [▶ 22]	Windows Systemzeit-Struktur in File-Time konvertieren
SYSTEMTIME_TO_STRING [▶ 22]	Windows Systemzeit-Struktur in String konvertieren
STRING_TO_SYSTEMTIME [▶ 23]	String in Windows Systemzeit-Struktur konvertieren
FILETIME_TO_DT [▶ 24]	Windows File-Time in DATE_AND_TIME konvertieren
FILETIME_TO_SYSTEMTIME [▶ 24]	Windows File-Time in Systemzeit-Struktur konvertieren
DEC_TO_BCD [▶ 211]	Dezimal-Zahlen in BCD-Zahlen konvertieren
BCD_TO_DEC [▶ 212]	BCD-Zahlen in Dezimal-Zahlen konvertieren
DEG_TO_RAD [▶ 26]	Grad-Winkel in Bogenmaß konvertieren
RAD_TO_DEG [▶ 26]	Bogenmaß in Grad-Winkel konvertieren
TIME_TO_OTSTRUCT [▶ 25]	TIME-Variable in eine Struktur mit aufgelösten Millisekunden, Sekunden, Minuten usw. konvertieren.
OTSTRUCT_TO_TIME [▶ 25]	Eine Struktur mit aufgelösten Millisekunden, Sekunden, Minuten usw. in TIME-Variable konvertieren.
F_SwapRealEx [▶ 53]	Tauscht um das Hi- und Lo-Word einer REAL-Variablen
BYTEARR_TO_MAXSTRING [▶ 47]	Konvertiert Byte Array in einen String
MAXSTRING_TO_BYTEARR [▶ 47]	Konvertiert String in ein Byte Array
F_TranslateFileTimeBias [▶ 61]	Konvertiert UTC-Zeit in Lokalzeit und umgekehrt (by bias)
FB_TzSpecificLocalTimeToFileTime [▶ 185]	Konvertiert kontinuierliche Lokalzeit (file time format) in UTC-Zeit
FB_TzSpecificLocalTimeToSystemTime [▶ 183]	Konvertiert kontinuierliche Lokalzeit (structured system time format) in UTC-Zeit
FB_FileTimeToTzSpecificLocalTime [▶ 187]	Konvertiert UTC-Zeit (file time format) in Lokalzeit
FB_SystemTimeToTzSpecificLocalTime [▶ 189]	Konvertiert UTC-Zeit (structured system time format) in Lokalzeit

String-Formatfunktionen

Name	Beschreibung
LREAL_TO_FMTSTR [▶ 45]	Konvertiert eine Fliesskommazahl in einen String mit der gewünschten Anzahl der Nachkommastellen.
DWORD_TO_DECSTR [▶ 35]	Konvertiert Dezimalzahl in einen Dezimalstring.
DWORD_TO_HEXSTR [▶ 37]	Konvertiert Dezimalzahl in einen Hexadezimalstring.

Name	Beschreibung
DWORD_TO_OCTSTR [▶ 38]	Konvertiert Dezimalzahl in einen Oktalstring.
DWORD_TO_BINSTR [▶ 36]	Konvertiert Dezimalzahl in einen Binärstring.
DINT_TO_DECSTR [▶ 44]	Konvertiert vorzeichenbehaftete Dezimalzahl in einen Dezimalstring.
F_FormatArgToStr [▶ 54]	Konvertiert und formatiert eine Dezimalzahl oder eine Fließkommazahl in einen String.
FB_FormatString [▶ 156]	Konvertiert und formatiert bis zu 10 Argumente (Dezimal- oder Fließkommazahlen).
FB_EnumStringNumbers [▶ 161]	Durchsucht einen String nach Zahlen.
F_ToUCase [▶ 62]	Konvertiert Kleinbuchstaben in Grossbuchstaben in einem String.
F_ToLCase [▶ 63]	Konvertiert Grossbuchstaben in Kleinbuchstaben in einem String.
F_LTrim [▶ 63]	Entfernt Leerzeichen am Anfang eines Strings.
F_RTrim [▶ 64]	Entfernt Leerzeichen am Ende eines Strings.
DATA_TO_HEXSTR [▶ 48]	Konvertiert Binärdaten in Hexadezimal-String.
HEXSTR_TO_DATA [▶ 49]	Konvertiert Hexadezimal-String in Binärdaten.

64 bit Funktionen (unsigned)

Name	Beschreibung
ULARGE_INTEGER [▶ 79]	Initialisiert/setzt eine 64 bit Zahl
UInt64Add64 [▶ 79]	Addiert zwei 64 bit Zahlen
UInt64Add64Ex [▶ 80]	Addiert zwei 64 bit Zahlen (mit Overflow check)
UInt64Sub64 [▶ 80]	Subtrahiert zwei 64 bit Zahlen
UInt64Cmp64 [▶ 84]	Vergleicht zwei 64 bit Zahlen
UInt32x32To64 [▶ 81]	Multipliziert zwei 32 bit Zahlen. Das Ergebnis ist eine 64 bit Zahl
UInt64Mul64 [▶ 81]	Multipliziert zwei 64 bit Zahlen. Das Ergebnis ist eine 64 bit Zahl
UInt64Mul64Ex [▶ 82]	Multipliziert zwei 64 bit Zahlen. Das Ergebnis ist eine 64 bit Zahl (mit Overflow check)
UInt64Div64 [▶ 82]	Division zweier 64 bit Zahlen.
UInt64Div64Ex [▶ 83]	Division zweier 64 bit Zahlen (mit Restergebnis).
UInt64Mod64 [▶ 84]	Modulo-Division zweier 64 bit Zahlen
UInt64And [▶ 85]	Bitweise AND zweier 64 bit Zahlen
UInt64Or [▶ 85]	Bitweise OR zweier 64 bit Zahlen
UInt64Not [▶ 86]	Bitweise NOT einer 64 bit Zahl
UInt64Xor [▶ 85]	Bitweise XOR zweier 64 bit Zahlen
UInt64Rol [▶ 88]	Bitweise Linksrotation einer 64 bit Zahl
UInt64Ror [▶ 89]	Bitweise Rechtsrotation einer 64 bit Zahl
UInt64Shl [▶ 88]	Bitweises Links-Shift einer 64 bit Zahl
UInt64Shr [▶ 88]	Bitweises Rechts-Shift einer 64 bit Zahl
UInt64Min [▶ 86]	Minimumfunktion
UInt64Max [▶ 87]	Maximumfunktion
UInt64Limit [▶ 87]	Limitierung
UInt64isZero [▶ 89]	Prüft ob der Wert der 64 bit Zahl ist Null
UINT64_TO_STRING [▶ 91]	Konvertiert 64 bit Zahl in einen STRING
UINT64_TO_LREAL [▶ 91]	Konvertiert 64 bit Zahl in einen LREAL
STRING_TO_UINT64 [▶ 90]	Konvertiert einen STRING in eine 64 bit Zahl

Name	Beschreibung
LREAL_TO_UINT64 [▶ 90]	Konvertiert LREAL in eine 64 bit Zahl

64 bit Funktionen (signed)

Name	Beschreibung
LARGE_INTEGER [▶ 91]	Initialisiert/setzt eine 64 bit Zahl
Int64Add64 [▶ 92]	Addiert zwei 64 bit Zahlen
Int64Add64Ex [▶ 92]	Addiert zwei 64 bit Zahlen (mit Overflow check)
Int64Sub64 [▶ 93]	Subtrahiert zwei 64 bit Zahlen
Int64Cmp64 [▶ 94]	Vergleicht zwei 64 bit Zahlen
Int64Div64Ex [▶ 93]	Division zweier 64 bit Zahlen (mit Restergebnis)
Int64Not [▶ 94]	Bitweise NOT einer 64 bit Zahl
Int64isZero [▶ 95]	Prüft ob der Wert der 64 bit Zahl ist Null
Int64Negate [▶ 95]	Negiert eine 64 bit Zahl
INT64_TO_LREAL [▶ 96]	Konvertiert 64 bit Zahl in einen LREAL
LREAL_TO_INT64 [▶ 95]	Konvertiert LREAL in eine 64 bit Zahl
LARGE_TO_ULARGE [▶ 96]	Konvertiert eine vorzeichenbehaftete 64 bit Zahl in eine vorzeichenlose 64 bit Zahl
ULARGE_TO_LARGE [▶ 96]	Konvertiert eine vorzeichenlose 64 bit Zahl in eine vorzeichenbehaftete 64 bit Zahl

16 Bit Festkommazahlen (signed)

Name	Beschreibung
FIX16Add [▶ 99]	Addiert zwei Festkommazahlen
FIX16Align [▶ 100]	Ändert die Auflösung einer Festkommazahl
FIX16Sub [▶ 102]	Subtrahiert zwei Festkommazahlen
FIX16Div [▶ 101]	Dividiert zwei Festkommazahlen
FIX16Mul [▶ 101]	Multipliziert zwei Festkommazahlen
LREAL_TO_FIX16 [▶ 97]	Konvertiert LREAL in eine Festkommazahl
WORD_TO_FIX16 [▶ 98]	Konvertiert WORD in eine Festkommazahl
FIX16_TO_LREAL [▶ 98]	Konvertiert eine Festkommazahl in LREAL
FIX16_TO_WORD [▶ 99]	Konvertiert eine Festkommazahl in WORD

Byte-Order-Konvertierungsfunktionen

Name	Beschreibung
HOST_TO_BE16 [▶ 121]	Host-To-Network Konvertierung (16 bit Zahl)
HOST_TO_BE32 [▶ 122]	Host-To-Network Konvertierung (32 bit Zahl)
HOST_TO_BE64 [▶ 122]	Host-To-Network Konvertierung (64 bit Zahl)
HOST_TO_BE128 [▶ 122]	Host-To-Network Konvertierung (128 bit Zahl)
BE16_TO_HOST [▶ 123]	Network-To-Host Konvertierung (16 bit Zahl)
BE32_TO_HOST [▶ 123]	Network-To-Host Konvertierung (32 bit Zahl)
BE64_TO_HOST [▶ 124]	Network-To-Host Konvertierung (64 bit Zahl)
BE128_TO_HOST [▶ 124]	Network-To-Host Konvertierung (128 bit Zahl)

Andere

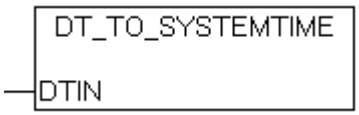
Name	Beschreibung
FB_BasicPID [▶ 152]	Einfacher PID controller
F_GetVersionTcUtilities [▶ 73]	Versionsinformationen der Bibliothek lesen
IsFinite [▶ 50]	Überprüft die Formatierung einer Gleitkommazahl nach der IEEE
F_YearIsLeapYear [▶ 59]	Ermittelt ob ein Jahr Schaltjahr ist
F_GetMaxMonthDays [▶ 57]	Ermittelt die maximale Anzahl der Monatstage
F_GetDOYOfYearMonthDay [▶ 57]	Ermittelt die Nummer des Tages im Jahr
F_GetMonthOfDOY [▶ 58]	Ermittelt den Monat anhand der Nummer des Tages im Jahr
F_GetDayOfWeek [▶ 65]	Ermittelt die Nummer des Wochentages.
F_GetWeekOfTheYear [▶ 65]	Ermittelt die Kalenderwoche
F_GetDayOfMonthEx [▶ 60]	Ermittelt das Datum des ersten, zweiten usw Wochentages in einem bestimmten Monat und Jahr
RTC [▶ 213]	"Software"-RTC (Real Time Clock)
RTC_EX [▶ 214]	"Software"-RTC (Real Time Clock)
RTC_EX2 [▶ 215]	"Software"-RTC (Real Time Clock)
FB_FileRingBuffer [▶ 166]	Schreibt/liest Datensätze in oder von der Datei (FIFO).
FB_MemRingBuffer [▶ 168]	Schreibt/liest Datensätze in oder von einer Puffervariable (FIFO).
FB_MemRingBufferEx [▶ 170]	Schreibt/liest Datensätze in oder von einer Puffervariable (FIFO).
FB_StringRingBuffer [▶ 176]	Schreibt/liest Strings in oder von einer Puffervariable (FIFO).
FB_MemStackBuffer [▶ 174]	Schreibt/liest Datensätze in oder von einer Puffervariable (LIFO).
FB_HashTableCtrl [▶ 191], F_CreateHashTableHnd [▶ 66]	Einfache Hash-Tabelle.
FB_LinkedListCtrl [▶ 192], F_CreateLinkedListHnd [▶ 67]	Einfache verkettete Liste (doppelt verkettet).
DCF77_TIME [▶ 222]	Ein einfacher DCF77-Dekoder.
DCF77_TIME_EX [▶ 225]	DCF77-Dekoder mit Plausibilitätsprüfung von zwei aufeinanderfolgenden Telegrammen und Zeitzoneninformation.

CSV-Format Hilfsbausteine

Name	Beschreibung
CSVFIELD_TO_STRING [▶ 67]	Konvertiert den Wert eines Strings mit einem Datenfeld im CSV-Format in eine SPS-Stringvariable
STRING_TO_CSVFIELD [▶ 70]	Konvertiert den Wert einer SPS-Stringvariablen in einen String mit einem Datenfeld im CSV-Format
CSVFIELD_TO_ARG [▶ 69]	Konvertiert einen Bytepuffer mit einem Datenfeld im CSV-Format in einen Wert einer beliebigen SPS-Variablen
ARG_TO_CSVFIELD [▶ 71]	Konvertiert den Wert einer beliebigen SPS-Variablen in einen Bytepuffer mit einem Datenfeld im CSV-Format
FB_CSVMemBufferReader [▶ 228]	Teilt Datensätze im CSV-Format die in einem Bytepuffer vorliegen in einzelne Datenfelder.
FB_CSVMemBufferWriter [▶ 229]	Erzeugt aus einzelnen Datenfeldern einzelne oder mehrere Datensätze in einem Bytepuffer

3 Funktionen

3.1 DT_TO_SYSTEMTIME



Mit der Funktion "DT_TO_SYSTEMTIME" kann eine im DATE_AND_TIME - Format (DT) SPS-Variable in eine Windows Systemzeit-Struktur konvertiert werden. Die Systemzeit hat eine Auflösung von 1ms, und das DATE_AND_TIME eine Auflösung von 1s. Die Variable "wMilliseconds" in der Systemzeit-Struktur liefert daher immer den Wert Null zurück.

FUNCTION DT_TO_SYSTEMTIME : TIMESTRUCT

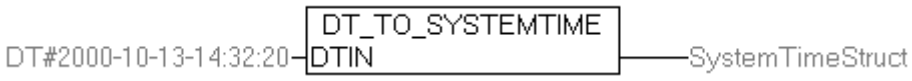
TIMESTRUCT [▶ 231]

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

DTIN:: Das zu konvertierende Datum und Uhrzeit in DATE_AND_TIME - Format.

Beispiel für einen Aufruf in FUP:

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



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.2 DT_TO_FILETIME

Mit der Funktion "DT_TO_FILETIME" kann eine im DATE_AND_TIME - Format (DT) SPS-Variable in das FILETIME-Format (64bit) konvertiert werden.

FUNCTION DT_TO_FILETIME : T_FILETIME

T_FILETIME [▶ 240]

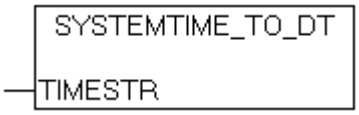
```
VAR_INPUT
    DTIN        : DT;
END_VAR
```

DTIN:: Das zu konvertierende Datum und Uhrzeit in DATE_AND_TIME - Format.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build >= 1030 TwinCAT v2.10.0 Build >= 1232	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.3 SYSTEMTIME_TO_DT



Mit der Funktion "SYSTEMTIME_TO_DT" kann die Windows Systemzeit-Struktur in das in der SPS gängige DATE_AND_TIME - Format (DT) konvertiert werden. Die Systemzeit hat eine Auflösung von 1ms, und das DATE_AND_TIME eine Auflösung von 1s. Die Millisekunden aus der Systemzeit werden bei der Konvertierung berücksichtigt und auf den DATE_AND_TIME - Rückgabewert entsprechend aufgerundet. Setzen Sie das wMilliseconds-Element in der Windows Systemzeit-Struktur auf Null um das Aufrunden zu deaktivieren.

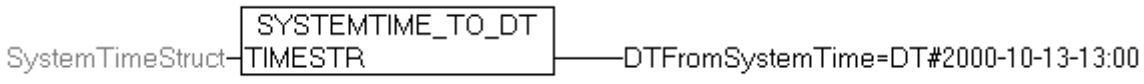
FUNCTION SYSTEMTIME_TO_DT : DT

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

TIMESTR: Struktur [▶ 231] mit der zu konvertierenden Windows Systemzeit..

Beispiel für einen Aufruf in FUP:

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

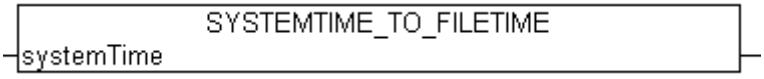


Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.4 SYSTEMTIME_TO_FILETIME



Mit der Funktion kann die Windows Systemzeit-Struktur in das Filetime-Format konvertiert werden. Der Wochentag wDayOfWeek der SystemTime-Variablen wird ignoriert. Das Systemzeit-Jahr muss größer als 1601 und kleiner als 30827 sein.

FUNCTION SYSTEMTIME_TO_FILETIME : T_FILETIME

T_FILETIME [▶ 240]

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

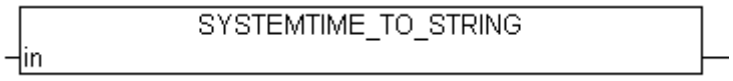
systemTime: Struktur [▶ 231] mit der zu konvertierenden Windows-Systemzeit.

Rückgabeparameter	Beschreibung
0	Fehler, falscher SystemTime Parameterwert.
> 0	Kein Fehler. File time.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.5 SYSTEMTIME_TO_STRING



Die Funktion konvertiert die Windows Systemzeit-Struktur in einen String mit folgendem Format: **YYYY-MM-DD-hh:mm:ss.xxx** :

- YYYY: Jahr (1601..9999)
- MM: Monat (01..12)
- DD: Tag (01..31)
- hh: Stunde (00..23)
- mm: Minuten (00..59)
- ss: Sekunden (00..59)
- xxx: Millisekunde (000..999)

FUNCTION SYSTEMTIME_TO_STRING : STRING(24)

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

in: Struktur [▶ 231] mit der zu konvertierenden Windows-Systemzeit.

Beispiel 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 ) );
```

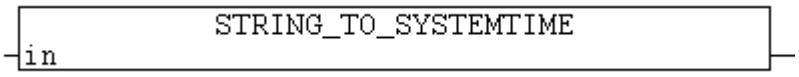
Onlineansicht:

```
0001 ⊞ fbGetSystemTime
0002 ⊞ fileTime
0003     sTime = '2005-08-29-07:28:00.355'
0004
0005
0006

0001 fbGetSystemTime(timeLoDW=>fileTime.dwLowDateTime, timeHiDW=>fileTime.dwHighDateTime );
0002 sTime :=SYSTEMTIME_TO_STRING( FILETIME_TO_SYSTEMTIME( fileTime ) );
0003
0004
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1241	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.6 STRING_TO_SYSTEMTIME



Die Funktion konvertiert einen String in den Windows SYSTEMTIME -Zeitformat.

FUNCTION STRING_TO_SYSTEMTIME: TIMESTRUCT

TIMESTRUCT [▶ 231]

VAR_INPUT

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

in:

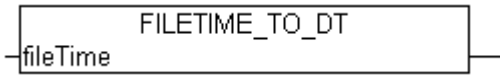
Der zu konvertierende String muss folgenden Format haben: **'YYYY-MM-DD-hh:mm:ss.xxx'**

- YYYY: Jahr (1601..9999);
- MM: Monat (01..12);
- DD: Tag (01..31);
- hh: Stunde (00..23);
- mm: Minute (00..59);
- ss: Sekunde (00..59);
- xxx: Millisekunde (000..999);

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1310 oder höher	PC or CX (x86, ARM)	TcUtilities.Lib

3.7 FILETIME_TO_DT



Die Funktion "FILETIME_TO_DT" konvertiert die Zeit im FILETIME-Format in das DATE_AND_TIME-Format (DT). Das DT-Format hat einen kleineren Wertebereich als das FILETIME-Format und nur eine Sekundengenauigkeit. Aus diesem Grund wird der zu konvertierende FILETIME-Wert limitiert. Das zulässige Minimum entspricht dem *DT#1970-01-01-00:00:00* und das Maximum dem *DT#2106-02-06-06:28:15* Wert.

FUNCTION FILETIME_TO_DT : DT

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

fileTime: Die zu konvertierende Zeit im FILETIME-Format [▶ 240].

Beispiel in ST:

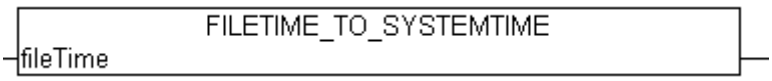
```
PROGRAM MAIN
VAR
    fbSystemTime    : GETSYSTEMTIME;
    timeAsFileTime  : T_FILETIME;
    timeAsDT        : DT;
END_VAR
```

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 und höher TwinCAT v2.10.0 Build > 1240 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.8 FILETIME_TO_SYSTEMTIME



Die Funktion "FILETIME_TO_SYSTEMTIME" konvertiert die Zeit im FILETIME-Format in das "lesbare" SYSTEMTIME-Format. Die Konvertierung schlägt fehl, wenn das höchstwertige Bit der 64 bit fileTime-Variablen gesetzt ist. Die TIMESTRUCT-Membervariablen haben in diesem Fall den Wert Null.

FUNCTION FILETIME_TO_SYSTEMTIME: TIMESTRUCT

TIMESTRUCT [▶ 231]

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

fileTime:: Die zu konvertierende Zeit im FILETIME-Format [▶ 240]

Beispiel in ST:

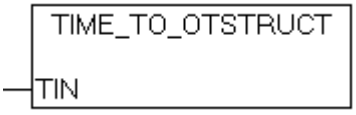
```
PROGRAM MAIN
VAR
    fbSystemTime      : GETSYSTEMTIME;
    timeAsFileTime    : T_FILETIME;
    timeAsSystemTime  : TIMESTRUCT;
END_VAR
```

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 und höher TwinCAT v2.10.0 Build > 1240 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.9 TIME_TO_OTSTRUCT



Mit der Funktion "TIME_TO_OTSTRUCT" kann eine TIME-Konstante oder Variable in eine Struktur mit den aufgelösten Millisekunden, Sekunden, Minuten, Stunden, Tagen und Wochen konvertiert werden.

FUNCTION TIME_TO_OTSTRUCT : OTSTRUCT

OTSTRUCT [▶ 231]

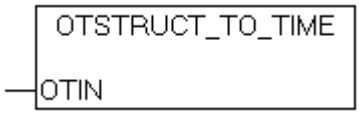
```
VAR_INPUT
    TIN             :TIME;
END_VAR
```

TIN: Die zu konvertierende TIME-Variable.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.10 OTSTRUCT_TO_TIME



Mit der Funktion "OTSTRUCT_TO_TIME" kann eine Struktur mit den aufgelösten Millisekunden, Sekunden, Minuten, Stunden, Tagen und Wochen in eine TIME-Variable konvertiert werden.

FUNCTION OTSTRUCT_TO_TIME : TIME

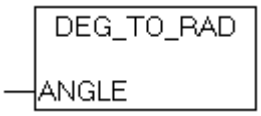
```
VAR_INPUT
    OTIN        : OTSTRUCT;
END_VAR
```

OTIN: Die zu konvertierende Struktur [▶ 231].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.11 DEG_TO_RAD



Die Funktion konvertiert einen Grad-Winkel in Bogenmaß.

FUNCTION DEG_TO_RAD : LREAL

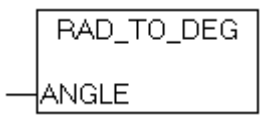
```
VAR_INPUT
    ANGLE       : LREAL;
END_VAR
```

ANGLE:: Der zu konvertierende Winkel in Grad.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.12 RAD_TO_DEG



Die Funktion konvertiert das Bogenmaß in einen Grad-Winkel.

FUNCTION RAD_TO_DEG : LREAL

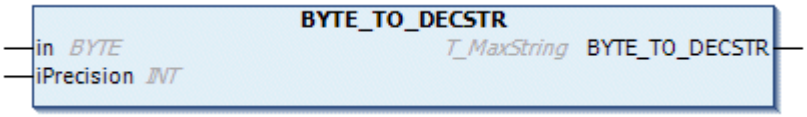
```
VAR_INPUT
    ANGLE:      LREAL;
END_VAR
```

ANGLE :: Das zu konvertierende Bogenmaß.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.13 BYTE_TO_DECSTR



Die Funktion konvertiert eine Dezimalzahl in einen Dezimalstring (Basis 10).

FUNCTION BYTE_TO_DECSTR : T_MaxString

T_MaxString

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

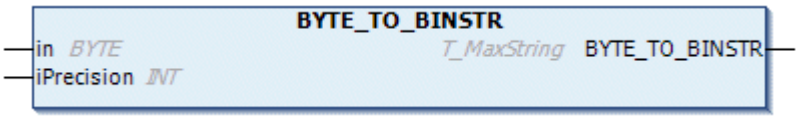
in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.14 BYTE_TO_BINSTR



Die Funktion konvertiert eine Dezimalzahl in einen Binärstring (Basis 2).

FUNCTION BYTE_TO_BINSTR : T_MaxString

T_MaxString

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

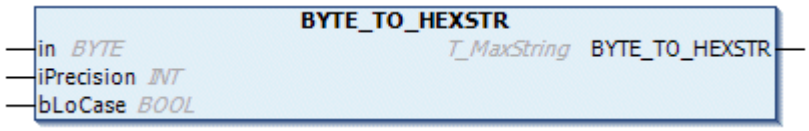
in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.15 BYTE_TO_HEXSTR



Die Funktion konvertiert eine Dezimalzahl in einen Hexadezimalstring (Basis 16).

FUNCTION BYTE_TO_HEXSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

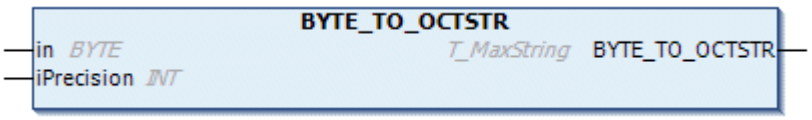
iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

bLoCase: Dieser Parameter bestimmt ob Klein- oder Großbuchstaben bei der Konvertierung benutzt werden sollen. FALSE => "ABCDEF", TRUE => "abcdef".

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.16 BYTE_TO_OCTSTR



Die Funktion konvertiert eine Dezimalzahl in einen Oktalstring (Basis 8).

FUNCTION BYTE_TO_OCTSTR : T_MaxString

T_MaxString

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

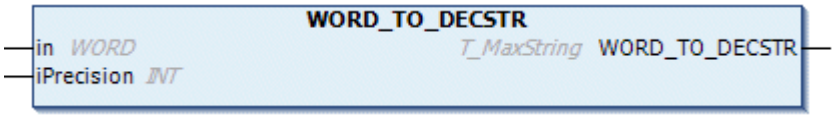
in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.17 WORD_TO_DECSTR



Die Funktion konvertiert eine Dezimalzahl in einen Dezimalstring (Basis 10).

FUNCTION WORD_TO_DECSTR : T_MaxString

T_MaxString

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

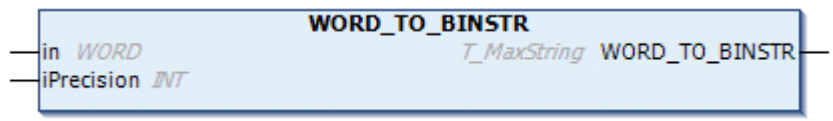
in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.18 WORD_TO_BINSTR



Die Funktion konvertiert eine Dezimalzahl in einen Binärstring (Basis 2).

FUNCTION WORD_TO_BINSTR : T_MaxString

T_MaxString

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

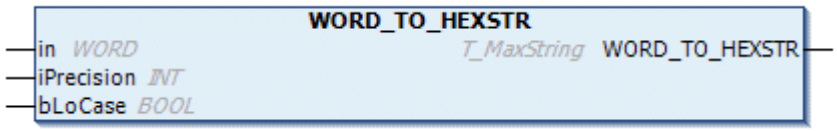
in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.19 WORD_TO_HEXSTR



Die Funktion konvertiert eine Dezimalzahl in einen Hexadezimalstring (Basis 16).

FUNCTION WORD_TO_HEXSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

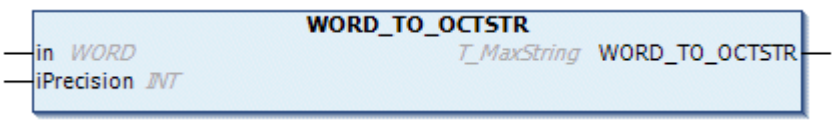
iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

bLoCase: Dieser Parameter bestimmt ob Klein- oder Großbuchstaben bei der Konvertierung benutzt werden sollen. FALSE => "ABCDEF", TRUE => "abcdef".

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.20 WORD_TO_OCTSTR



Die Funktion konvertiert eine Dezimalzahl in einen Oktalstring (Basis 8).

FUNCTION WORD_TO_OCTSTR : T_MaxString

T_MaxString

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

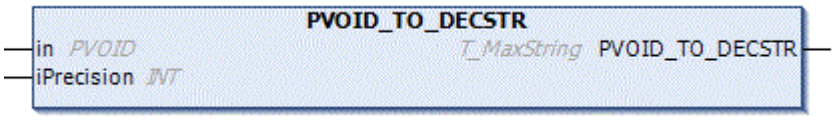
in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.21 PVOID_TO_DECSTR



Die Funktion konvertiert den Wert einer Pointervariablen vom Typ PVOID in einen Dezimalstring (Basis 10).

FUNCTION PVOID_TO_DECSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Pointervariable.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Beispiel:

```
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 );
```

Das Ergebnis:

s1 = ''

s2 = '0'

s3 = '0000000000000000'

s4 = '2279473749' (kann variieren)

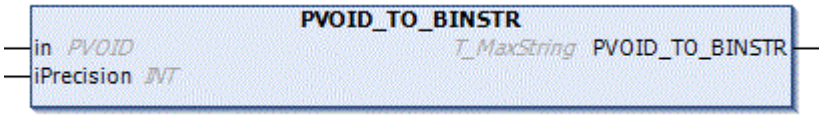
s5 = '2279473749' (kann variieren)

s6 = '0000002279473749' (kann variieren)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.22 PVOID_TO_BINSTR



Die Funktion konvertiert den Wert einer Pointervariablen vom Typ PVOID in einen Binärstring (Basis 2).

FUNCTION PVOID_TO_BINSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Pointervariable.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Beispiel:

```
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_BINSTR( pCnt, 32 );
pCnt := ADR( nCnt );
s4   := PVOID_TO_BINSTR( pCnt, 0 );
s5   := PVOID_TO_BINSTR( pCnt, 1 );
s6   := PVOID_TO_BINSTR( pCnt, 32 );
```

Das Ergebnis:

s1 = ''

s2 = '0'

s3 = '00000000000000000000000000000000'

s4 = '10000111110111100000001001010101' (kann variieren)

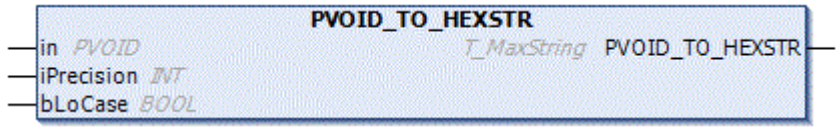
s5 = '10000111110111100000001001010101' (kann variieren)

s6 = '10000111110111100000001001010101' (kann variieren)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.23 PVOID_TO_HEXSTR



Die Funktion konvertiert den Wert einer Pointervariablen vom Typ PVOID in einen Hexadezimalstring (Basis 16).

FUNCTION PVOID_TO_HEXSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Pointervariable.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

bLoCase: Dieser Parameter bestimmt, ob Klein- oder Großbuchstaben bei der Konvertierung benutzt werden sollen. FALSE => "ABCDEF", TRUE => "abcdef".

Beispiel:

```
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 );
```

Das Ergebnis:

s1 = ''

s2 = ''

s3 = '0'

s4 = '0'

s5 = '0000000000000000'

s6 = '0000000000000000'

s7 = '87CBC255' (kann variieren)

s8 = '87cbc255' (kann variieren)

s9 = '87CBC255' (kann variieren)

s10 = '87cbc255' (kann variieren)

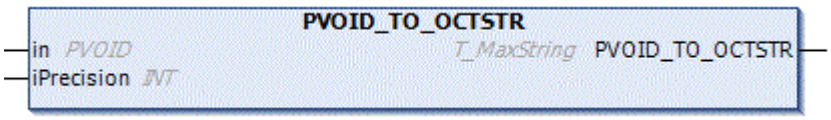
s11 = '0000000087CBC255' (kann variieren)

s12 = '0000000087cbc255' (kann variieren)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.24 PVOID_TO_OCTSTR



Die Funktion konvertiert den Wert einer Pointervariablen vom Typ PVOID in einen Oktalstring (Basis 8).

FUNCTION PVOID_TO_OCTSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Pointervariable.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Beispiel:

```
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 );
```

Das Ergebnis:

s1 = ''

s2 = '0'

s3 = '0000000000000000'

s4 = '20767501125' (kann variieren)

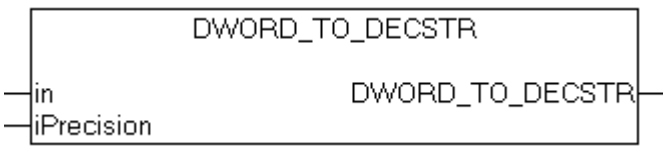
s5 = '20767501125' (kann variieren)

s6 = '0000020767501125' (kann variieren)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.25 DWORD_TO_DECSTR



Die Funktion konvertiert eine Dezimalzahl in einen Dezimalstring (Basis 10).

FUNCTION DWORD_TO_DECSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Beispiel 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 );
```

Das Ergebnis:

s1 = '43981'

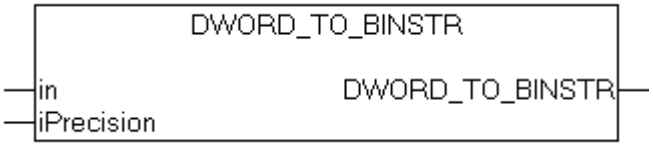
s2 = '0000043981'

s3 = ''

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.26 DWORD_TO_BINSTR



Die Funktion konvertiert eine Dezimalzahl in einen Binärstring (Basis 2).

FUNCTION DWORD_TO_BINSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Beispiel 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 );
```

Das Ergebnis:

s1 = '0000000010000001'

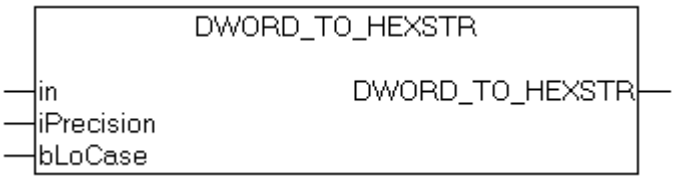
s2 = '1111'

s3 = ''

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.27 DWORD_TO_HEXSTR



Die Funktion konvertiert eine Dezimalzahl in einen Hexadezimalstring (Basis 16).

FUNCTION DWORD_TO_HEXSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

bLoCase: Dieser Parameter bestimmt, ob Klein- oder Großbuchstaben bei der Konvertierung benutzt werden sollen. FALSE => "ABCDEF", TRUE => "abcdef".

Beispiel 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 );
```

Das Ergebnis:

s1 = 'ABCD'

s2 = 'abcd'

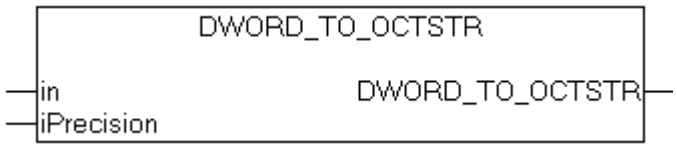
s3 = '000F'

s4 = ''

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.28 DWORD_TO_OCTSTR



Die Funktion konvertiert eine Dezimalzahl in einen Oktalstring (Basis 8).

FUNCTION DWORD_TO_OCTSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (Digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring.

Beispiel 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 );
```

Das Ergebnis:

s1 = '125715'

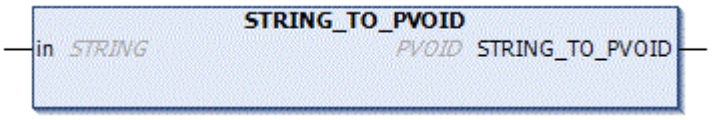
s2 = '0000125715'

s3 = ''

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.29 STRING_TO_PVOID



Die Funktion konvertiert eine Stringvariable in eine Pointervariable vom Typ PVOID. Die Funktion liefert den Rückgabewert Null, wenn der Eingangsstring fehlerhafte Zeichen enthält und nicht als Adresse interpretiert werden kann.

FUNCTION STRING_TO_PVOID: PVOID

VAR_INPUT

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

in: Die zu konvertierende Stringvariable

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

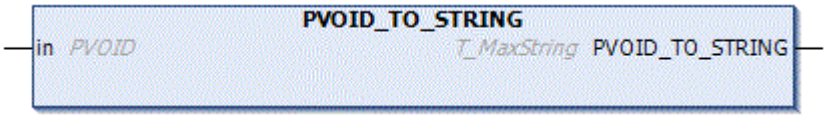
pP4 = 2301908600

pP5 = 0

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.30 PVOID_TO_STRING



Die Funktion konvertiert den Wert einer Pointervariablen vom Typ PVOID in einen Hexadezimalstring (Basis 16). Der Hexadezimalstring besitzt den SPS-Prefix: '16#'. Die Auflösung ist fix und beträgt 8 Stellen auf einem 32 Bit System.

FUNCTION PVOID_TO_STRING : T_MaxString

T_MaxString

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

in: Die zu konvertierende Pointervariable.

Beispiel:

```
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 );
```

Das Ergebnis auf einem 32 bit System:

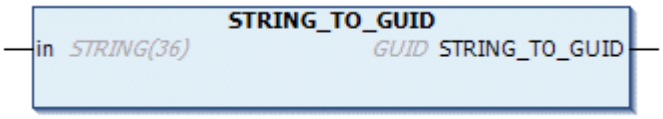
s1 = '16#00000000'

s2 = "16#87DE0255' (kann variieren)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.31 STRING_TO_GUID



Diese Funktion konvertiert eine GUID-Stringvariable (ohne geschweifte Klammern) in eine strukturierte GUID [▶ 249]-Variable.

FUNCTION STRING_TO_GUID: GUID

GUID [▶ 249]

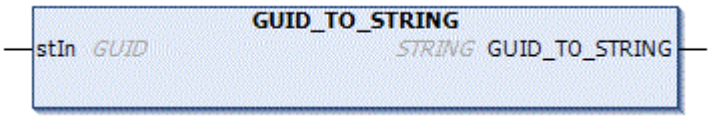
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.32 GUID_TO_STRING



Diese Funktion konvertiert eine strukturierte GUID [▶ 249]-Variable in eine GUID-Stringvariable (ohne geschweifte Klammern).

FUNCTION GUID_TO_STRING : STRING

VAR_INPUT

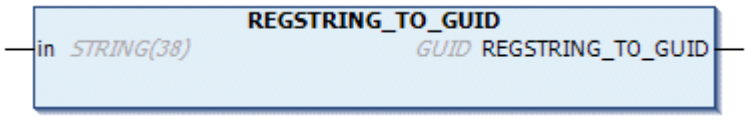
```
VAR_INPUT
      stIn : GUID;
END_VAR
```

Rückgabewert	Bedeutung
'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'	Kein Fehler (‘x’ ist ein Hexadezimal-Halbbyte)
00000000-0000-0000-0000-000000000000'	Kein Fehler, GUID hat den Initialwert (alle Bytes sind Null)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.33 REGSTRING_TO_GUID



Diese Funktion konvertiert eine Registry-GUID-Stringvariable (eingeschlossen in geschweifte Klammern) in eine strukturierte GUID [▶ 249]-Variable.

FUNCTION REGSTRING_TO_GUID: GUID

GUID [▶ 249]

VAR_INPUT

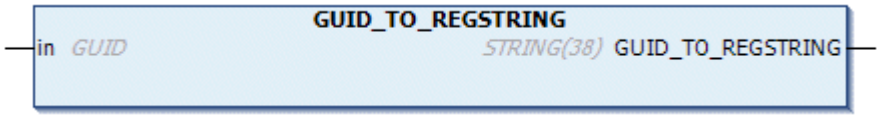
```
VAR_INPUT
      in : STRING(38);
END_VAR
```

Rückgabewert	Bedeutung
'{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'	Kein Fehler (‚x' ist ein Hexadezimal-Halbbyte).
'{00000000-0000-0000-0000-000000000000}'	Konvertierung fehlgeschlagen oder GUID hat den Initialwert (alle Bytes sind Null)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.34 GUID_TO_REGSTRING



Diese Funktion konvertiert eine strukturierte GUID [▶ 249]-Variable in eine Registry-GUID-Stringvariable (eingeschlossen in geschweifte Klammern).

FUNCTION GUID_TO_REGSTRING: STRING(38)

VAR_INPUT

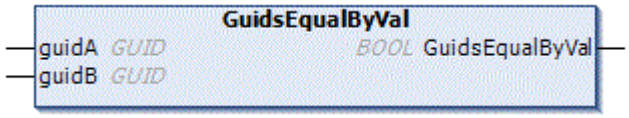
```
VAR_INPUT
      in : GUID;
END_VAR
```

Rückgabewert	Bedeutung
'{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'	Kein Fehler (‘x’ ist ein Hexadezimal-Halbbyte)
'{00000000-0000-0000-0000-000000000000}'	Kein Fehler, GUID hat den Initialwert (alle Bytes sind Null)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.35 GuidsEqualByVal



Diese Funktion vergleicht zwei GUID Werte.

FUNCTION GuidsEqualByVal: BOOL

VAR_INPUT

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

Rückgabewert	Bedeutung
FALSE	guidA <> guidB
TRUE	guidA = guidB

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.36 HEXASCNIBBLE_TO_BYTE

Diese Funktion konvertiert den ASCII-Code eines hexadezimalen Halbbytezeichens in den Dezimalwert.

FUNCTION HEXASCNIBBLE_TO_BYTE : BYTE

VAR_INPUT

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

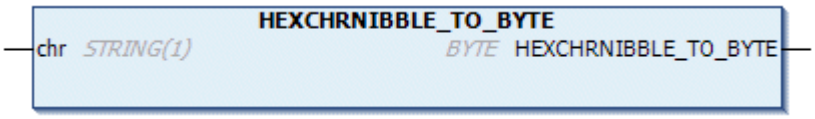
asc: Ascii-Code eines hexadezimalen Halbbytezeichens (Ascii code von: ‚0' bis ‚9' oder ‚a' bis ‚f' oder ‚A' bis ‚F').

Rückgabewert	Bedeutung
0 bis 15	Erfolgreich, kein Fehler.
255	Fehler, fehlerhafter Eingangsparameterwert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.37 HEXCHRNIBBLE_TO_BYTE



Diese Funktion konvertiert ein hexadezimales Halbbytezeichen in seinen dezimalen Wert.

FUNCTION HEXCHRNIBBLE_TO_BYTE : BYTE

VAR_INPUT

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

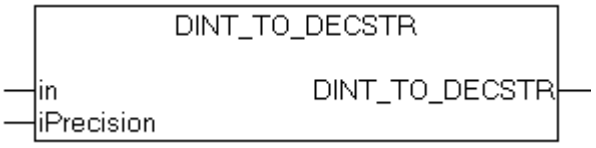
chr: Hexadezimales Halbbyte-Zeichen (‚0' bis ‚9' oder ‚a' bis ‚f' oder ‚A' bis ‚F').

Rückgabewert	Bedeutung
0 bis 15	Erfolgreich, kein Fehler.
255	Fehler, falscher Eingangsparameterwert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.38 DINT_TO_DECSTR



Die Funktion konvertiert eine vorzeichenbehaftete Dezimalzahl in einen Dezimalstring (Basis 10).

FUNCTION DINT_TO_DECSTR : T_MaxString

T_MaxString

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

in: Die zu konvertierende Dezimalzahl.

iPrecision: Minimale Anzahl der erscheinenden Stellen (digits). Ist die tatsächliche Anzahl der signifikanten Stellen kleiner als der *iPrecision*-Parameter, dann wird der resultierende String links mit Nullen aufgefüllt. Ist die Anzahl der signifikanten Stellen größer ist als der *iPrecision*-Parameter, dann wird der resultierende String nicht abgeschnitten! Hat der *iPrecision*-Parameter und der *in*-Parameter den Wert Null, dann ist der resultierende String ein Leerstring. Bei negativen Zahlen erscheint im resultierenden String zusätzlich das negative Vorzeichen.

Beispiel 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 );
```

Das Ergebnis:

s1 = '-1234

s2 = '-0000001234'

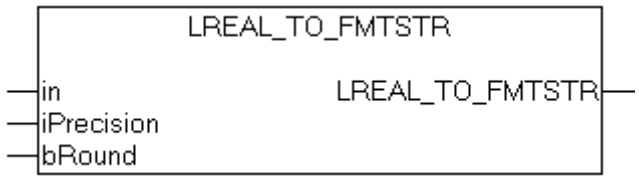
s3 = ''

s4 = '0000001234'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.39 LREAL_TO_FMTSTR



Die Funktion konvertiert und formatiert eine Fließkommazahl in eine String-Variable mit dem folgenden Format: **[–]dddd.dddd** (dddd sind Dezimalzahlen). Die Anzahl der Zahlen vor dem Dezimalpunkt hängt von dem Wert der Fließkommazahl ab. Die Anzahl der Zahlen hinter dem Dezimalpunkt hängt von der geforderten Präzision ab. Ein Vorzeichen erscheint nur bei negativen Werten. Bei einem unendlichen positiven Wert wird **'#INF'** und bei unendlichem negativen Wert: **'-#INF'** zurückgeliefert. Wenn die übergebene Variable einen unzulässigen Wert besitzt (NaN, Not-a-Number), dann wird **'#QNAN'** oder **'-#QNAN'** zurückgeliefert. Wenn die Länge des formatierten Strings die maximal zulässige Länge des resultierenden Strings überschreitet, dann wird **'#OVF'** oder **'-#OVF'** zurückgeliefert.

FUNCTION LREAL_TO_FMTSTR : STRING(510)

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

in: Fließkommazahl die konvertiert und formatiert werden soll.

iPrecision: Präzision. Der Wert bestimmt die Anzahl der Zahlen hinter dem Dezimalpunkt. Bei dem Minimalwert (Null) erscheinen keine Nachkommastellen, der Maximalwert von *iPrecision* wird durch die Anzahl der Zahlen vor dem Dezimalpunkt und die maximal zulässige Länge des resultierenden Strings begrenzt. Wenn *in* = 0 und *iPrecision* = 0 dann wird String '0' zurückgeliefert.

bRound: Beim gesetzten *bRound*-Parameter wird der formatierte String auf die entsprechenden Anzahl der Nachkommastellen (*iPrecision*) gerundet. Beim Runden gilt folgende Regel: Hat die Dezimalzahl hinter der letzten gewünschten Nachkommastelle den Wert >= 5 dann wird aufgerundet sonst nicht.

Beispiel:

Die Zahl 0.46523 soll in einen String mit zwei Nachkommstellen konvertiert und gerundet werden.

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

Das Ergebnis ist: '0.47';

Bedingt durch die interne Darstellung der Fließkommazahlen und Rundungsfehler während der Konvertierung kann es vorkommen, dass der resultierende String nicht exakt dem Wert der *in*-Variable entspricht. Die maximale Anzahl der signifikanten Dezimalstellen bei den LREAL-Variablen ist auf 15-Stellen begrenzt.

Beispiel:

```
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 );
```

Das Ergebnis ist:

s1 = '0.4999999999999999756000000' Dies ist die interne Darstellung der *double*-Variablen. Diese Zahl wird als Ausgangspunkt für die Rundungsoperation benutzt.

s2 = '0.49'

Durch die Rundung ergeben sich dann folgende Ergebnisse:

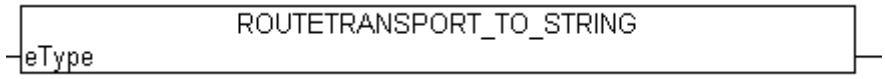
s3 = '0'

s4 = '0.50'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
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



Die Funktion konvertiert die AMS Message Router Transportschichtkennung in einen String.

FUNCTION ROUTETRANSPORT_TO_STRING : STRING

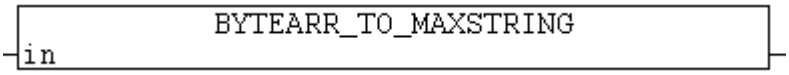
```
VAR_INPUT
    eType   : E_RouteTransportType;
END_VAR
```

eType:: Die zu konvertierende Transportschichtkennung [▶ 236]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1033 und höher TwinCAT v2.10.0 Build > 1257 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.41 BYTEARR_TO_MAXSTRING



Konvertiert die einzelnen ASCII-Codes eines Byte-Arrays in einen String.

FUNCTION BYTEARR_TO_MAXSTRING : T_MaxString

T_MaxString

VAR_INPUT

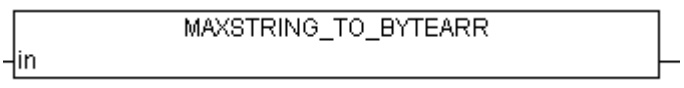
```
VAR_INPUT
    in : ARRAY[0..MAX_STRING_LENGTH] OF BYTE;
END_VAR
```

in: Byte-Array variable (MAX_STRING_LENGTH default value: 255).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86, ARM)	TcUtilities.Lib

3.42 MAXSTRING_TO_BYTEARR



Konvertiert einen String in einzelne ASCII-Codes eines Byte-Arrays.

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

VAR_INPUT

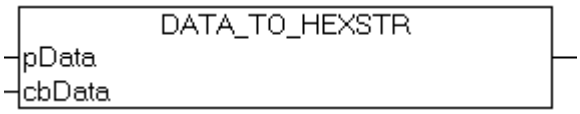
```
VAR_INPUT
    in : T_MaxString;
END_VAR
```

in: String der konvertiert werden soll. (T_MaxString)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.43 DATA_TO_HEXSTR



Die Funktion konvertiert Binärdaten in einen Hexadezimalstring. Mit dieser Funktion können einfache Datentypen und Struktur-Variablen konvertiert werden. Die maximale Länge der Binärdaten darf aber 85 Byte nicht überschreiten! Beim Überschreiten der maximalen Länge wird dem Ergebnisstring ein Punkt-Zeichen hinzugefügt ('.') und die Konvertierung abgebrochen. Die verbleibenden Datenbytes werden nicht mehr konvertiert. Bei fehlerhaften Funktionsparametern (*pData* = Null oder *cbData* = Null) liefert die Funktion ein Leerstring als Ergebnis.

FUNCTION DATA_TO_HEXSTR : T_MaxString

T_MaxString

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

pData: Anfangsadresse (Pointer) auf die zu konvertierenden Binärdaten. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbData: Max. Länge der zu konvertierenden Binärdaten. Die Länge darf 85 Bytes nicht überschreiten! Die Länge kann mit dem SIZEOF-Operator ermittelt werden.

bLoCase: Dieser Parameter legt fest, ob Groß- oder Kleinbuchstaben bei der Konvertierung verwendet werden sollen. TRUE = Kleinbuchstaben, FALSE = Großbuchstaben.

Beispiel in ST:

Bitte beachten Sie, dass die Datengröße der *overflow*-Variablen die 85 Bytes überschreitet. Der Ergebnisstring *sH5* wird aus diesem Grund mit einem Punkt abgeschlossen.

Bei der *number*-Variablen ist die Bytefolge vertauscht, weil die Speicherorganisation der Zählervariablen dem Little-Endian-Format entspricht (auch Intel-Format genannt).

```
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 );
```

Das Ergebnis:

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'

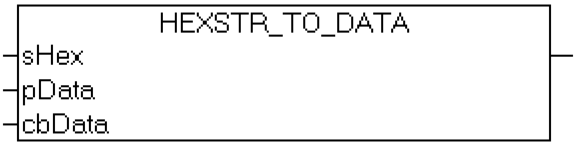
sH4 = '00 00 00 00'

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.'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1331	PC or CX (x86, ARM)	TcUtilities.Lib

3.44 HEXSTR_TO_DATA



Die Funktion konvertiert einen Hexadezimalstring in Binärdaten und liefert die Anzahl der erfolgreich konvertierten Datenbytes als Ergebnis zurück. Es dürfen nur Leerzeichen als Trennzeichen in dem zu konvertierenden Hexadezimalstring verwendet werden. Bei den Hex-Zeichen sind Klein- und Großbuchstaben zugelassen. Bei einem Fehler oder unerlaubten Zeichen wird die Konvertierung abgebrochen und eine Null-Länge als Ergebnis zurückgeliefert.

FUNCTION HEXSTR_TO_DATA : UDINT

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

sHex: Der zu konvertierende Hexadezimalstring. Z.B.: 'AB CD 01 23'

pData: Anfangsadresse (Pointer) auf den Zielpuffer in den die konvertieren Datenbytes geschrieben werden sollen. Die Adresse kann mit dem ADR-Operator ermittelt werden.

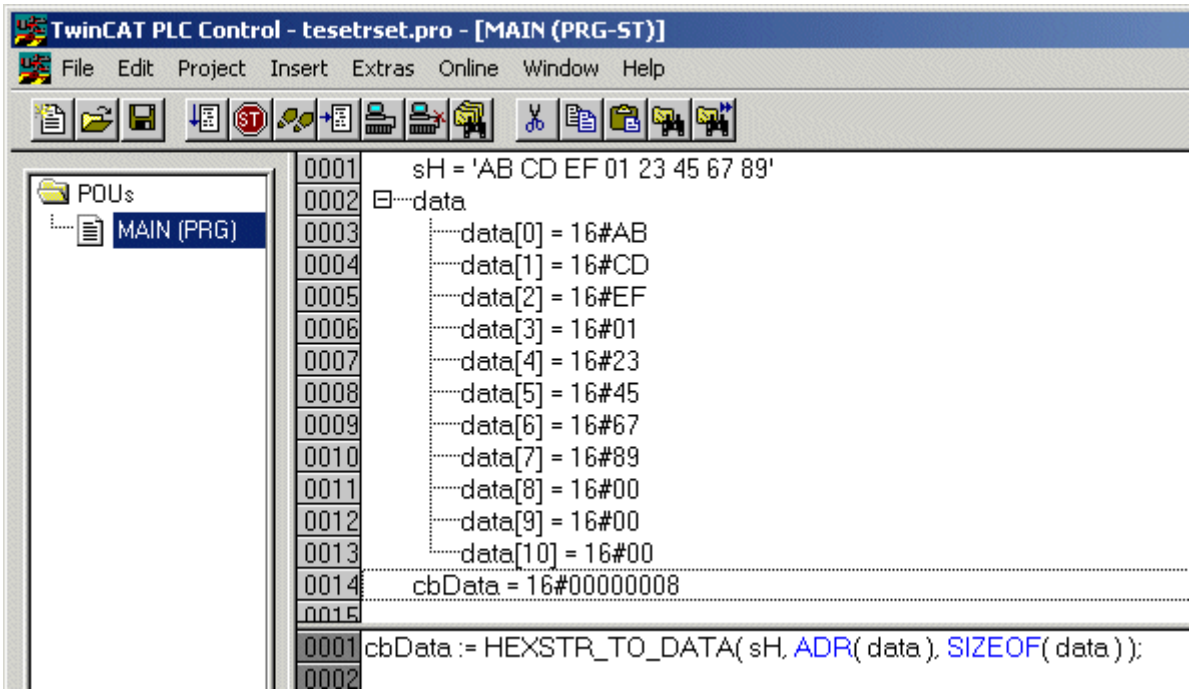
cbData: Max. verfügbare Länge des Zielpuffers. Die Länge kann mit dem SIZEOF-Operator ermittelt werden.

Beispiel 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 ) );
```

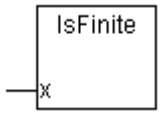
Das Ergebnis (Online):



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1334	PC or CX (x86, ARM)	TcUtilities.Lib

3.45 IsFinite



Die Funktion IsFinite() liefert TRUE zurück, wenn deren Argument einen endlichen Wert besitzt (INF < x < +INF). D.h. die Funktion liefert FALSE zurück, wenn der Argument unendlich oder NaN ist (NaN = Not a number). IsFinite() überprüft, ob die Formatierung einer LREAL oder REAL-Variablen der IEEE entspricht.

INF-Zahlen können auf einem Laufzeitsystem dann vorkommen, wenn das Ergebnis einer mathematischen Operation den darstellbaren Bereich überschreitet oder unterschreitet. Z.B.:

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

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

NaN-Zahlen können im Laufzeitsystem dann vorkommen, wenn deren eigentliche Formatierung (Speicherinhalte) durch unerlaubten Zugriff (z.B. durch Benutzung der MEMCPY, MEMSET Funktionen) überschrieben wurden. Z.B.:

```
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 *)
```

Beim Aufruf einer Konvertierungsfunktion mit einer NaN oder INF-Zahl als Parameter wird auf einem PC-System (i368) eine FPU-Exception ausgelöst. Diese Exception führt anschließend zum Stopp der SPS. Mit der Funktion IsFinite() kann der Wert der Variablen überprüft, die FPU-Exception vermieden und die Programmausführung fortgesetzt werden.

FUNCTION IsFinite : BOOL

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

x: Eine Hilfsstruktur [▶ 239] mit Informationen zu der zu überprüfenden REAL oder LREAL-Variablen. Die Strukturparameter müssen beim Aufruf von IsFinite() mit Hilfsfunktionen F_REAL [▶ 103] oder F_LREAL [▶ 103] erzeugt und als Parameter übergeben werden.

Beispiel für einen Aufruf in ST:

Im folgenden Beispiel wird die Formatierung einer REAL- und einer LREAL-Variablen überprüft und eine FPU-Exception vermieden.

```
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
```

Im folgenden Fall kann eine FPU-Exception durch Überprüfung mit IsFinite() nicht vermieden werden:

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

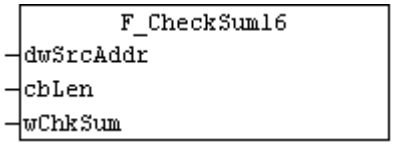
```
IF IsFinite( F_LREAL( bigFloat ) ) THEN
    smallDigit := LREAL_TO_INT( bigFloat );
END_IF
```

Die bigFloat-Variable besitzt zwar richtige Formatierung, der Variablenwert ist aber zu groß um diesen in einen INT-Typ konvertieren zu können. Auf einem PC-System (i368) wird eine Exception ausgelöst und das Laufzeitsystem gestoppt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.46 F_CheckSum16



Mit der Funktion "F_CheckSum16" kann eine 16-Bit-Checksumme von beliebigen Daten ermittelt werden.

FUNCTION F_CheckSum16 : WORD

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

dwSrcAddr: Adresse des Datenpuffers.

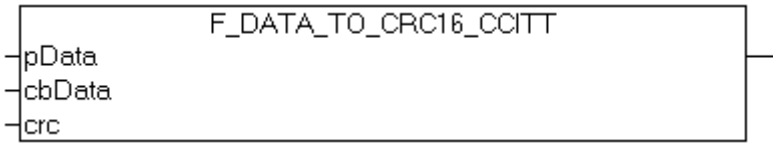
cbLen: Länge des Datenpuffers.

wChkSum: Initialwert = 0 oder letzte Checksumme.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 735	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.47 F_DATA_TO_CRC16_CCITT



Mit der Funktion "F_DATA_TO_CRC16_CCITT" kann eine 16-Bit-CRC-CCITT (zyklische Redundanzprüfung) von beliebigen Daten ermittelt werden. Intern wird die Funktion F_BYTE_TO_CRC16_CCITT [▶ 53] benutzt.

Weitere Informationen zum benutzten Algorithmus finden Sie in der Dokumentation der F_BYTE_TO_CRC16_CCITT [▶ 53]-Funktion.

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: Adresse des Datenpuffers.

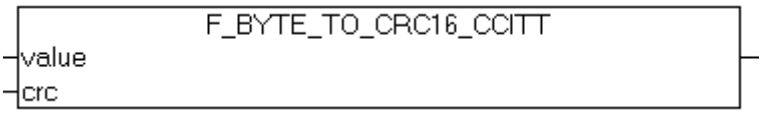
cbData: Länge des Datenpuffers.

crc: Initialwert = 16#FFFF oder 16#0000 oder der letzte CRC.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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



Mit der Funktion "F_BYTE_TO_CRC16_CCITT" kann eine 16-Bit-CRC-CCITT (zyklische Redundanzprüfung) von einzelnen Datenbytes ermittelt werden.

Der benutzte generator Polynom: Name : CRC-16 CCITT

- Standard : CRC-CCITT
- Referencen : 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
```

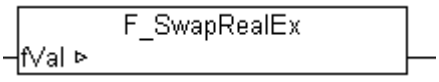
value: Der zu konvertierende Datenbyte.

crc: Initialwert = 16#FFFF oder 16#0000 oder der letzte CRC.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1524	PC or CX (x86, ARM)	TcUtilities.Lib

3.49 F_SwapRealEx



Die Speicherdarstellung einer REAL-Zahl auf einem Buscontroller (165) unterscheidet sich von der Speicherdarstellung einer REAL-Zahl auf einem Intel-System (PC).
Um eine REAL-Zahl eines Buscontrollers auf einem PC richtig darstellen zu können müssen die Hi- und Lo-Words der REAL-Zahl vertauscht werden. Die Programmierumgebung macht dies bereits im Online- oder Simulations-Mode. Um die REAL-Daten eines Buscontrollers über das Netzwerk (ADS-Protokoll, ADSDLL, AdsOcx usw.) anzufordern und auf einem Intel-PC richtig darzustellen, müssen die REAL-Daten in das richtige Format konvertiert werden. Dieses kann auf der Buscontroller- oder PC-Seite erfolgen.

Mit der Funktion F_SwapReal können die REAL-Variablen (die z.B. von einer VB-Applikation eingelesen werden oder mit TwinCAT Scope View aufgezeichnet werden sollen) auf der PC-Seite in das passende Format konvertiert werden. Die Funktion verändert die Speicherdarstellung des übergebenen *fVal*-Parameters (VAR_IN_OUT).

FUNCTION F_SwapRealEx : BOOL

VAR_IN_OUT

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

fVal: Der zu konvertierende REAL Wert.

Rückgabeparameter	Bedeutung
TRUE	Kein Fehler
FALSE	Fehler bei der Funktionsausführung

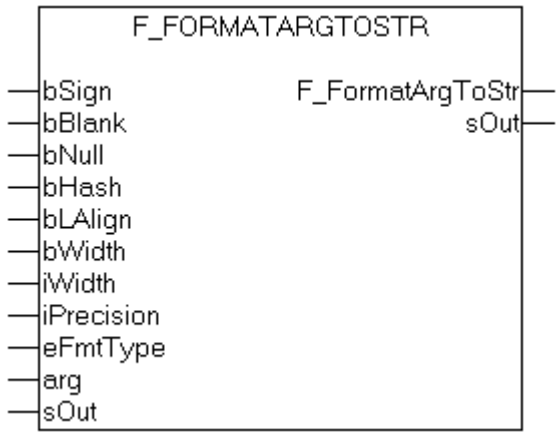
Bespiel:

Siehe: Beispiel: Kommunikation BC/BX<->PC/CX (F_SwapRealEx). [▶ 252]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10 Build > 1301	PC or CX (x86) CX (ARM)	TcUtilities.Lib

3.50 F_FormatArgToStr



Format-Hilfsfunktion. Diese Funktion wird intern von dem FB_FormatString [▶ 156]-Funktionsbaustein benutzt. Mit dieser Funktion kann eine Variable vom Typ T_Arg [▶ 239] entsprechend der Formatspezifikation [▶ 275] in einen formatierten String konvertiert werden.

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: Das Vorzeichen-Flag [▶ 276].

bBlank: Das Leerzeichen-Flag [▶ 276].

bNull: Das Null-Flag [▶ 276].

bHash: Das Hash-Prefix-Flag [▶ 276].

bLAlign: Das Ausrichtungs-Flag [▶ 276] (TRUE=left align).

bWidth: Wenn TRUE, dann wird der iWidth-Parameter ausgewertet, sonst nicht.

iWidth: Width [▶ 277]-Parameter.

iPrecision: Precision [▶ 277]-Parameter.

eFmtType: Type [▶ 275]-Parameter.

arg: Das zu formatierende Argument. Folgende Hilfsfunktionen können benutzt werden um SPS-Variablen unterschiedlichsten Typs in den benötigten Datentyp T_Arg [▶ 239] zu konvertieren: 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: Beim Erfolg liefert diese Variable den formatierten Ausgabestring.

Rückgabeparameter	Bedeutung
0	Kein Fehler
<> 0	Fehler. Fehlerbeschreibung finden Sie unter: Format Fehlercodes [▶ 250]

Beispiele in ST:

Formatieren einer BYTE-Variablen als Binärstring.

```
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 );
```

Das Ergebnis:

s1 = '10000000'

s2 = ' 10000000'

s3 = '10000000 '

s4 = '2#10000000 '

L1 = 8

L2 = 20

L3 = 20

L4 = 20

Formatieren einer LREAL-Variablen.

```
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 );
```

Das Ergebnis:

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

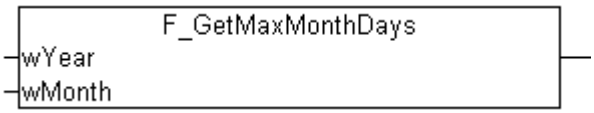
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

- E_TypeFieldParam [▶ 236]
- F_BYTE [▶ 104]
- F_LREAL [▶ 103]

3.51 F_GetMaxMonthDays



Die Funktion liefert die maximale Anzahl der Monatstage in einem bestimmten Monat und Jahr.

FUNCTION F_GetMaxMonthDays : WORD

VAR_INPUT

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

wYear: Jahr.

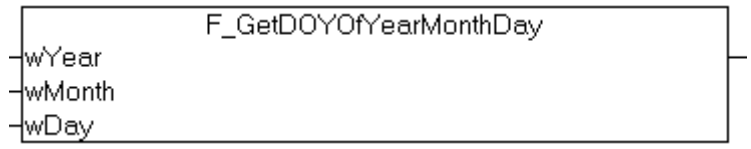
wMonth: Monat (1 bis 12).

Rückgabeparameter	Beschreibung
0	Fehler, falscher wMonth Parameterwert
> 0	Kein Fehler. Maximale Anzahl der Monatstage.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.52 F_GetDOYOfYearMonthDay



Die Funktion berechnet die Nummer des Tages im Jahr.

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: Jahr (0 ~ 2999).

wMonth: Monat (1 ~ 12).

wDay: Tag (1 ~ 31).

Rückgabeparameter	Beschreibung
0	Fehler, falscher wYear, wMonth oder wDay Parameterwert
> 0	Kein Fehler. Nummer des Tages im Jahr (1 ~ 366)

Beispiel:

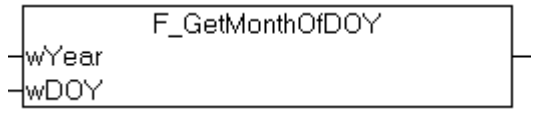
```
PROGRAM P_TEST_DOY
VAR
    wYear   : WORD;
    wDOY    : WORD;
    wMonth  : WORD;
    wDay    : WORD;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1530	PC or CX (x86, ARM)	TcUtilities.Lib

3.53 F_GetMonthOfDOY



Die Funktion berechnet den Monat anhand der Nummer des Tages im Jahr.

FUNCTION F_GetMonthOfDOY: WORD

VAR_INPUT

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

wYear: Jahr (0 ~ 2999).

wDOY: Nummer des Tages im vorgegebenen Jahr dessen Monat ermittelt werden soll (1 ~ 366).

Rückgabeparameter	Beschreibung
0	Fehler, falscher wYear oder wDOY Parameterwert.
> 0	Kein Fehler. Monat (1 ~ 12).

Beispiel:

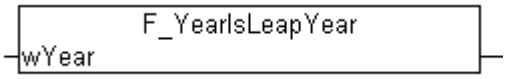
```
PROGRAM P_TEST_DOY
VAR
    wYear       : WORD;
    wDOY        : WORD;
    wMonth      : WORD;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1340 TwinCAT v2.11.0 Build > 1530	PC or CX (x86, ARM)	TcUtilities.Lib

3.54 F_YearIsLeapYear



Die Funktion ermittelt ob ein Jahr ein Schaltjahr ist.

FUNCTION F_YearIsLeapYear: BOOL

VAR_INPUT

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

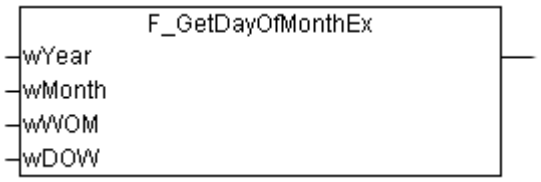
wYear: Jahr.

Rückgabeparameter	Beschreibung
TRUE	Schaltjahr
FALSE	Kein Schaltjahr

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.55 F_GetDayOfMonthEx



Die Funktion berechnet das Datum des ersten, zweiten usw. Wochentages in einem bestimmten Monat und Jahr (z..B. das Datum des zweiten Montags in Januar 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: Jahr (1601 bis 30827).

wMonth: Monat (1 bis 12).

wWOM: Woche im Monat (1 bis 5). Der Wert 1 entspricht der ersten Woche, 2 der zweiten Woche und 5 der letzten Woche (auch wenn der Monat keine 5 Wochen besitzt).

wDOW: Wochentag (0 bis 6). 0 = Sonntag, 1 = Montag... 6 = Samstag.

Rückgabeparameter	Beschreibung
0	Fehler, falscher oder unzulässiger Funktionsparameter
> 0	Kein Fehler. Monatstag

Beispiel:

Das Beispiel ermittelt das Datum des zweiten Montags im August 2011. Das Ergebnis ist: 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 );
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

3.56 F_TranslateFileTimeBias

Diese Funktion rechnet die Eingangszeit mit Hilfe der angegebenen Bias-Zeitverschiebung in die Zeit einer anderen Zeitzone um. Mit dieser Funktion kann z.B. die Lokalzeit in UTC-Zeit (Universal Time Coordinates) und umgekehrt umgerechnet werden.

FUNCTION F_TranslateFileTimeBias: T_FILETIME

T_FILETIME [▶ 240]

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: Eingangszeit [▶ 240] die umgerechnet werden soll.

bias: Differenz zwischen der UTC-Zeit und der Lokalzeit in Minuten (positive oder negative Werte sind zulässig).

toUTC: Über diesen Parameter kann die Richtung angegeben werden in die die Eingangszeit konvertiert werden soll.

toUTC	Direction	Internal formula
FALSE	UTC -> Lokalzeit	Lokalzeit := UTC - Bias
TRUE	Lokalzeit -> UTC	UTC := Lokalzeit + Bias

Beispiel:

Die *in*-Variable enthält die zu konvertierende Zeit. Die *bToUTC*-Variable bestimmt die Konvertierungsrichtung. Bei *bToUTC* = TRUE wird die Lokalzeit in UTC-Zeit konvertiert und bei *bToUTC* = FALSE die UTC-Zeit in Lokalzeit. Die *WEST_EUROPE_TZI*-Konstante enthält die Zeitzoneninformationen für Westeuropa. Der benötigte Bias-Wert wird aus der Zeitzoneninformation in der Konstanten und der aktuellen bDST-Einstellung (Daylight Saving Time) errechnet. Die aktuelle Zeitzoneninformation eines TwinCAT-Systems kann alternativ mit dem Funktionsbaustein: FB_GetTimeZoneInformation [▶ 177] ermittelt werden.

Der Datentyp DT als Eingangszeit wurde nur wegen der Möglichkeit der optischen Kontrolle im Online-Mode gewählt. Konvertierungen in einem anderen Zeitformat sind nur bedingt zu empfehlen da die Konvertierungsfunktionen sehr rechenintensiv sein können.

```
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 ) );
```

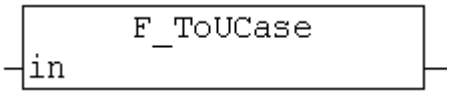
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias, FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

3.57 F_ToUCase



Die F_ToUCase-Funktion wandelt alle Kleinbuchstaben einer Zeichenfolge in Großbuchstaben um.

Eingeschränkter Zeichensatz

Standardmäßig verwendet die Konvertierungsfunktion den Zeichensatz der Windows Code Page 1252 Latin 1, SBCS (Single Byte Character Set). Ein anderer Zeichensatz kann zur Laufzeit (zur Zeit aber nur Windows Code Page 1250 Central European) über die globale Variable GLOBAL_SBCS_TABLE angewählt werden (siehe Beispiel).

FUNCTION F_ToUCase : T_MaxString

T_MaxString

VAR_INPUT

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

in: Der zu konvertierende String.

Beispiel:

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

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

Das Ergebnis der Konvertierung ist: 'TO UPPER CASE 1234567890 ÄÖÜß'

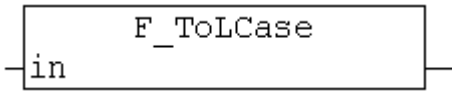
```
GLOBAL_SBCS_TABLE := eSBCS_CentralEuropean;
sUCase := F_ToUCase( 'to upper case 1234567890 ąęśćżźłó' );
```

Das Ergebnis der Konvertierung ist: 'TO UPPER CASE 1234567890 ĄĘŚĆŻŹŁÓ'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.58 F_ToLCase



Die F_ToLCase-Funktion wandelt alle Großbuchstaben einer Zeichenfolge in Kleinbuchstaben um.

Eingeschränkter Zeichensatz

Standardmäßig verwendet die Konvertierungsfunktion den Zeichensatz der Windows Code Page 1252 Latin 1, SBCS (Single Byte Character Set). Ein anderer Zeichensatz kann zur Laufzeit (zur Zeit aber nur Windows Code Page 1250 Central European) über die globale Variable GLOBAL_SBCS_TABLE angewählt werden (siehe Beispiel).

FUNCTION F_ToLCase : T_MaxString

T_MaxString

VAR_INPUT

```
VAR_INPUT
    in      : T_MaxString
END_VAR
```

in: Der zu konvertierende String.

Beispiel:

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

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

Das Ergebnis der Konvertierung ist: 'to lower case 1234567890 äöüß'

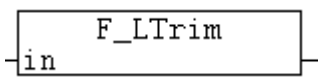
```
GLOBAL_SBCS_TABLE := eSBCS_CentralEuropean;
sLCase := F_ToLCase( 'TO LOWER CASE 1234567890 ĄĘŚĆŻŹŁÓ' );
```

Das Ergebnis der Konvertierung ist: 'to lower case 1234567890 ąęśćżźłó'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.59 F_LTrim



Entfernt von der Zeichenfolge führende Leerzeichen und gibt die reduzierte Zeichenfolge zurück.

FUNCTION F_LTrim : T_MaxString

T_MaxString

VAR_INPUT

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

Beispiel:

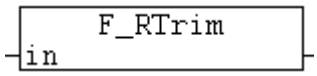
```
PROGRAM MAIN
VAR
    sLTrim : STRING;
END_VAR
```

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.60 F_RTrim



Schneidet vom angegebenen Wert alle Leerzeichen am Ende ab und gibt das Ergebnis zurück.

FUNCTION F_RTrim : T_MaxString

T_MaxString

VAR_INPUT

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

Beispiel:

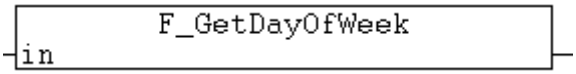
```
PROGRAM MAIN
VAR
    sRTrim : STRING;
    sLRTrim : STRING;
END_VAR
```

```
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>' *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

3.61 F_GetDayOfWeek



Die Funktion liefert die Nummer des Wochentags nach DIN 1355 / ISO 8601. Nach dieser Norm sind die Wochentage wie folgt Nummeriert: Montag = 1, Dienstag = 2, ... Sonntag = 7.

FUNCTION F_GetDayOfWeek : WORD

VAR_INPUT

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

in: Das Datum, dessen Wochentagsnummer ermittelt werden soll.

Beispiel in ST:

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

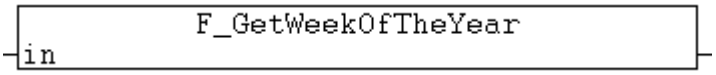
```
dayOfWeek := F_GetDayOfWeek(dtFirst);
```

Das Ergebnis ist 2 (Dienstag)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1325	PC or CX (x86, ARM)	TcUtilities.Lib

3.62 F_GetWeekOfTheYear



Die Funktion liefert die Nummer der Kalenderwoche zu einem vorgegebenen Datum nach der DIN 1355 / ISO 8601 Norm.

- Die erste Kalenderwoche ist definiert als die erste Woche, in die **mindestens vier Tage des neuen Jahres fallen (DIN 1355 / ISO 8601)**;
- Die Kalenderwochen beginnen Montags und jede Kalenderwoche enthält 7 Tage;
- Es gibt jährlich entweder 52 oder 53 Kalenderwochen. Es gibt 53 Kalenderwochen wenn ein Jahr mit einem Donnerstag beginnt oder endet;
- Die erste Kalenderwoche hat die Nummer 1;
- Der 29, 30 und 31 Dezember kann auch schon zur ersten Kalenderwoche des Folgejahres gehören;
- Der 1, 2 und 3 Januar kann auch noch zur letzten Kalenderwoche des Vorjahres gehören;

FUNCTION F_GetWeekOfTheYear : WORD

VAR_INPUT

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

in: Das Datum, dessen Kalenderwoche ermittelt werden soll.

Beispiel in ST:

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

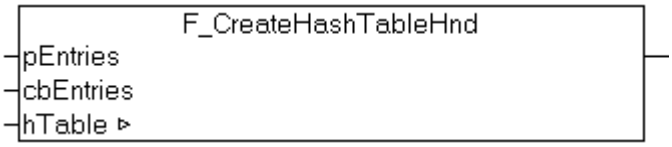
```
weekOfYear := F_GetWeekOfTheYear(dtNow);
```

Das Ergebnis ist 12.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1325	PC or CX (x86, ARM)	TcUtilities.Lib

3.63 F_CreateHashTableHnd



Die Funktion Initialisiert das Hash-Table-Handle. Das Tabellen-Handle muss einmalig durch Aufruf der F_CreateHashTableHnd-Funktion initialisiert werden.

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: Adresse von dem ersten T_HashTableEntry [▶ 243]-Arrayelement. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbEntries: Die T_HashTableEntry-Bytegröße . Die Bytegröße kann mit dem SIZEOF-Operator ermittelt werden.

VAR_IN_OUT

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

hTable: Hash-Tabellen-Handle [▶ 242] welches initialisiert werden soll. Das Handle wird bei den Zugriffen auf die Hash-Tabelle von dem Funktionsbaustein FB_HashTableCtrl [▶ 191] benötigt.

Rückgabeparameter	Beschreibung
TRUE	No error
FALSE	Error

Beispiel: In der Dokumentation FB_HashTableCtrl [▶ 191].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

3.64 F_CreateLinkedListHnd

Die Funktion Initialisiert das Linked-List-Handle. Das List-Handle muss einmalig durch Aufruf der *F_CreateLinkedListHnd*-Funktion initialisiert werden.

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: Adresse von dem ersten T_LinkedListEntry [▶ 243]-Arrayelement. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbEntries: Die Bytegröße des *T_LinkedListEntry*-Arrays. Die Bytegröße kann mit dem SIZEOF-Operator ermittelt werden.

VAR_IN_OUT

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

hList: Hash-Tabellen-Handle [▶ 243] welches initialisiert werden soll. Das Handle wird bei den Zugriffen auf die Liste von dem Funktionsbaustein FB_LinkedListCtrl [▶ 192] benötigt.

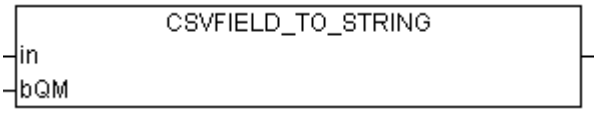
Rückgabeparameter	Beschreibung
TRUE	No error
FALSE	Error

Beispiel: Siehe in der Dokumentation von dem FB_LinkedListCtrl [▶ 192]-Funktionsbaustein.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcUtilities.Lib

3.65 CSVFIELD_TO_STRING



Die Funktion konvertiert ein Datenfeld im CSV-Datenfeldformat der als Quellstring vorliegt in einen Wert im SPS-Stringformat. Doppelte Einführungszeichen im Quellstring werden durch ein einfaches Einführungszeichen ersetzt. Beim gesetzten bQM-Parameter (QM = quotation marks) werden auch die äußeren (das Datenfeld einschließenden) Einführungszeichen aus dem Quellstring entfernt. Beim Erfolg

liefert die Funktion den konvertierten String als Ergebnis zurück. Die Funktion liefert einen Leerstring wenn bei der Konvertierung ein Fehler aufgetreten ist aber nur dann wenn der Quellstring auch kein Leerstring war.

Die Funktion wird normalerweise zusammen mit dem Funktionsbaustein FB_CSVMemBufferReader [▶ 228] verwendet um Datensätze im SPS-Speicher die im CSV-Format vorliegen zu lesen (zu interpretieren). Vorher werden die CSV-Datensätze meistens aus einer Datei in den SPS-Speicher eingelesen. Der Quellstring darf keine Binärdaten enthalten. Binärdaten mit dem Wert Null werden den String an der falschen Stelle terminieren und abschneiden! Um Datenfelder mit Binärdaten zu konvertieren verwenden Sie bitte die Funktion: CSVFIELD_TO_ARG [▶ 69].

FUNCTION CSFIELD_TO_STRING : T_MaxString

T_MaxString

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

in: Quellstring mit einem Datenfeld im CSV-Format der in einen Wert im SPS-Stringformat konvertiert werden soll.

bQM: Bei TRUE an diesem Eingang werden aus dem Quellstring auch die einschließenden Einführungszeichen entfernt.

bQM	Beschreibung	Quellstring	Ergebnisstring	CSV-Konform
FALSE	Ein Quellstring , der nicht mit den einschließenden Einführungszeichen versehen wird sollte möglichst nur aus Buchstaben und Zahlen bestehen. In diesem Fall darf der Quellstring keine nicht-druckbaren Steuerzeichen, Einführungszeichen, Semikolons, Kommas (US-CSV-Format) oder Binärdaten enthalten.			
		'Module_XA5'	'Module_XA5'	Ja
		'123456'	'123456'	Ja
		''	''	Ja
		'A"""B'	'A""B'	Nein
		'A""B'	'A"B'	Nein
		';'	';'	Nein
		'RN'	'RN'	Nein
		'AB$00CD'	'AB' (String wurde abgeschnitten)	Nein
TRUE	Ein Quellstring , der mit den einschließenden Einführungszeichen versehen wird darf auch die nicht-druckbaren Steuerzeichen, Einführungszeichen, Semikolons, Kommas (US-CSV-Format) enthalten. Binärdaten sind nicht zulässig.	'"Module_XA5"'	'Module_XA5'	Ja
		'"123456"'	'123456'	Ja
		'""'	''	Ja
		'"A"""B"'	'A""B'	Ja
		'"A""B"'	'A"B'	Ja
		'";"'	';'	Ja
		'"RN"'	'RN'	Ja
		'"AB$00CD"'	'AB' (String wurde abgeschnitten)	Nein

Beispiel 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 );
```

Das Ergebnis:

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

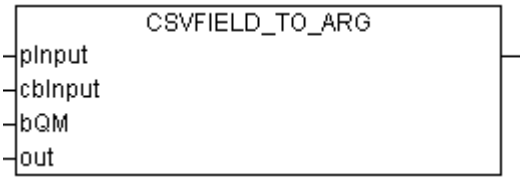
s2 = 'Module_50'

Weitere Informationen finden Sie hier: Beispiel: Schreiben/lesen einer CSV-Datei [▶ 270].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.66 CSVFIELD_TO_ARG



Die Funktion konvertiert den Wert von einem Datenfeld im CSV-Format das als Bytepuffer vorliegt in eine SPS-Variable. Doppelte Einführungszeichen im Datenfeld werden durch einfache Einführungszeichen ersetzt. Beim gesetzten bQM-Parameter (QM = quotation marks) werden auch die äußeren (das Datenfeld einschließende) Einführungszeichen aus den Eingangsdaten entfernt. Beim Erfolg liefert die Funktion die Länge der konvertierten Daten zurück. Beim Fehler und bei der Länge der Eingangsdaten Null liefert die Funktion den Wert Null. Die Applikation muss dafür sorgen das die SPS-Zielvariable groß genug ist um den Wert aufnehmen zu können.

Die Funktion wird normalerweise zusammen mit dem Funktionsbaustein FB_CSVMemBufferReader [▶ 228] verwendet um Datensätze im SPS-Speicher die im CSV-Format vorliegen zu lesen (parsen). Vorher werden die CSV-Datensätze meistens aus einer Datei in den SPS-Speicher eingelesen. Im Gegensatz zu der CSVFIELD_TO_STRING [▶ 67]-Funktion lassen sich mit dieser Funktion auch CSV-Datenfelder mit Binärdaten in SPS-Variablen konvertieren.

FUNCTION CSVFIELD_TO_ARG : UDINT

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

pInput: Anfangsadresse (Pointer) auf einen Bytepuffer mit dem zu konvertierenden Datenfeld im CSV-Format. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbInput: Die Länge des zu konvertierenden Datenfeldes in Byte. Die Länge kann mit dem SIZEOF-Operator ermittelt werden.

bQM: Bei TRUE an diesem Eingang werden aus den Felddaten auch die einschließenden Einführungszeichen entfernt.

out: SPS-Zielvariable in die der Wert des Datenfeldes [▶ 239] hineingeschrieben werden soll.

Beispiel in ST:

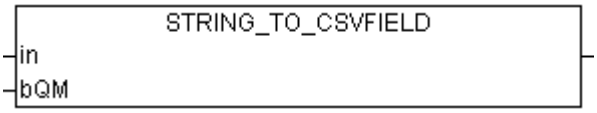
Siehe Beispiel in der Dokumentation des ARG_TO_CSVFIELD [▶ 71]-Funktionsbausteins.

Weitere Informationen finden Sie hier: Beispiel: Schreiben/lesen einer CSV-Datei [▶ 270].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.67 STRING_TO_CSVFIELD



Die Funktion konvertiert den Wert einer SPS-Stringvariablen in ein Datenfeld im CSV-Format als String. Einfache Einführungszeichen im Quellstring werden durch doppelte Einführungszeichen ersetzt. Beim gesetzten bQM-Parameter (QM = quotation marks) werden auch die äußeren (das CSV-Datenfeld einschließenden) Einführungszeichen hinzugefügt. Beim Erfolg liefert die Funktion den konvertierten String als Ergebnis zurück. Die Funktion liefert einen Leerstring wenn bei der Konvertierung ein Fehler aufgetreten ist aber nur dann wenn der Quellstring auch kein Leerstring war.

Die Funktion wird normalerweise zusammen mit dem Funktionsbaustein FB_CSVMemBufferWriter [▶ 229] verwendet um Datensätze im SPS-Speicher im CSV-Format zu erzeugen. Im nächsten Schritt kann der Speicherinhalt dann in die Datei geschrieben werden.

Der Quellstring darf keine Binärdaten enthalten. Binärdaten mit dem Wert Null würden den String an der falschen Stelle terminieren und abschneiden! Um Binärdaten zu konvertieren verwenden Sie bitte die Funktion: ARG_TO_CSVFIELD [▶ 71].

FUNCTION STRING_TO_CSVFIELD : T_MaxString

T_MaxString

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

in: Quellstring dessen Wert in ein Datenfeld im CSV-Format konvertiert werden soll.

bQM: Bei TRUE an diesem Eingang werden dem Ergebnisstring auch die einschließenden Einführungszeichen hinzugefügt.

bQM	Beschreibung	Quellstring	Ergebnisstring	CSV-Konform
FALSE	Quellstring , der nicht mit den einschließenden Einführungszeichen versehen wird sollte möglichst nur aus Buchstaben und Zahlen bestehen. In diesem Fall darf der Quellstring keine nicht-druckbaren Steuerzeichen, Einführungszeichen, Semikolon, Komma (US-CSV-Format) oder Binärdaten enthalten.			
		'Module_XA5'	'Module_XA5'	Ja
		'123456'	'123456'	Ja
		''	''	Ja
		'A""B'	'A""""B'	Nein
		'A"B'	'A""B'	Nein
		';'	';'	Nein
		'RN'	'RN'	Nein
		'AB$00CD'	'AB' (String wurde abgeschnitten)	Nein
TRUE	Quellstring , der mit den einschließenden Einführungszeichen versehen wird darf auch die nicht-druckbaren Steuerzeichen,	'Module_XA5'	'"Module_XA5"'	Ja
		'123456'	'"123456"'	Ja
		''	'""'	Ja
		'A""B'	'"A""""B"'	Ja
		'A"B'	'"A""B"'	Ja

bQM	Beschreibung	Quellstring	Ergebnisstring	CSV-Konform
	Einführungszeichen, Semikolon oder Komma (US-CSV-Format) enthalten. Binärdaten sind nicht zulässig.	';'	'";"'	Ja
		'RN'	'"RN"'	Ja
		'AB$00CD'	'"AB"' (String wurde abgeschnitten)	Nein

Beispiel 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 );
```

Das Ergebnis:

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

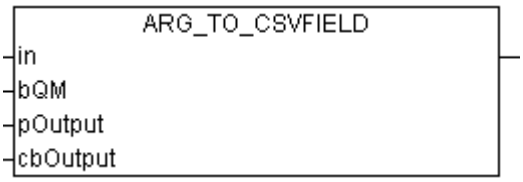
s2 = 'Module_50'

Weitere Informationen finden Sie hier: Beispiel: Schreiben/lesen einer CSV-Datei [▶ 270].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.68 ARG_TO_CSVFIELD



Die Funktion konvertiert den Wert einer SPS-Variablen in ein Datenfeld im CSV-Format. Einfache Einführungszeichen in den Quelldaten werden durch doppelte Einführungszeichen ersetzt. Beim gesetzten bQM-Parameter (QM = quotation marks) werden auch die äußeren (das CSV-Datenfeld einschließende) Einführungszeichen hinzugefügt. Beim Erfolg liefert die Funktion die Länge der konvertierten Daten als Ergebnis zurück. Die Funktion liefert Null wenn bei der Konvertierung Fehler aufgetreten sind oder bei fehlenden Daten. Das Ergebnis wird in den bereitgestellten Bytepuffer geschrieben. Die Applikation muss dafür sorgen, dass die Puffergröße so dimensioniert ist, damit das Ergebnis dort hineinpasst.

Die Funktion wird normalerweise zusammen mit dem Funktionsbaustein FB_CSVMemBufferWriter [▶ 229] verwendet, um Datensätze im SPS-Speicher im CSV-Format zu erzeugen. Im nächsten Schritt kann der Speicherinhalt dann in die Datei geschrieben werden. Im Gegensatz zu der STRING_TO_CSVFIELD [▶ 70]-Funktion lassen sich mit dieser Funktion auch SPS-Variablen mit Binärdaten in CSV-Datenfelder konvertieren.

FUNCTION ARG_TO_CSVFIELD : UDINT

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

in: SPS-Quellvariable deren Wert in ein Datenfeld [▶ 239] im CSV-Format konvertiert werden soll.

bQM: Bei TRUE an diesem Eingang werden die konvertierten Felddaten in Einführungszeichen eingeschlossen.

pOutput: Anfangsadresse (Pointer) auf den Ausgangspuffer. Die Pufferadresse kann mit dem ADR-Operator ermittelt werden. In diesen Puffer werden die Ergebnisdaten hineingeschrieben.

cbOutput: Die maximal verfügbare Größe des Ausgangspuffers in Byte. Die Länge des Ausgangspuffers kann mit dem SIZEOF-Operator ermittelt werden.

Beispiel in ST:

Im folgenden Beispiel wird gezeigt wie SPS-Variablen unterschiedlichen Typs in das CSV-Format und umgekehrt konvertiert werden können. Bei der ARG_TO_CSVFIELD-Konvertierung wird das Ergebnis in den Bytepuffer hineinkopiert (field1..field6). Bei der CSVFIELD_TO_ARG [▶ 69]-Konvertierung liegen die Quelldaten im Bytepuffer (field1..field6) und das Ergebnis wird in die TwinCAT SPS-Variable kopiert.

```
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 ) )
);
```

Das Ergebnis (Bytepuffer als hexadezimaler 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

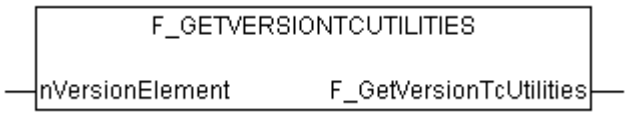
cbVar6 = 3

Weitere Informationen finden Sie hier: Beispiel: Schreiben/lesen einer CSV-Datei [▶ 270].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

3.69 F_GetVersionTcUtilities



Mit dieser Funktion können Versionsinformationen der SPS-Bibliothek ausgelesen werden.

FUNCTION F_GetVersionTcUtilities : UINT

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

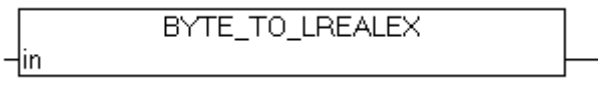
nVersionElement : Versionselement, das gelesen werden soll. Mögliche Parameter:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.70 BYTE_TO_LREALEX



In TwinCAT 2 auf ARM-Plattform wird die Konvertierung der vorzeichenlosen Zahlen in Fließkommezahlen von Typ: LREAL nicht unterstützt. Vorzeichenlose Zahlen mit einem gesetzten höchstwertigen Bit werden möglicherweise implizit in negative Fließkommazahlen konvertiert. Die hier beschriebene Funktion erlaubt in TwinCAT 2 eine explizite Konvertierung vom Typ BYTE in eine positive Fließkommazahl vom Typ LREAL (auch wenn das höchstwertige Bit gesetzt wurde und ohne Compiler-Warnung). In TwinCAT 3 werden vorzeichnlose Zahlen von Typ BYTE immer (implizit und explizit) in eine positive Fließkommazahl konvertiert.

FUNCTION BYTE_TO_LREALEX : LREAL

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

in: Die zu konvertierende Zahl.

Beispiel:

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

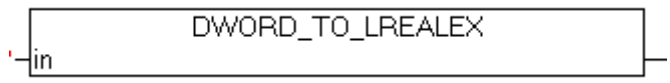
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nByte	+255, Warning 1105*	+255	+255
fLreal := BYTE_TO_LREAL(nByte)	+255, Warning 1105*	+255	+255
fLreal := BYTE#16#FF	+255, Warning 1105*	+255	+255
fLreal := 16#FF	+255	+255	+255
fLreal := BYTE_TO_LREALEX(nByte)	+255	+255	+255
fLreal := BYTE_TO_LREALEX(BYTE#16#FF)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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.71 DWORD_TO_LREALEX



In TwinCAT 2 auf ARM-Plattform wird die Konvertierung der vorzeichenlosen Zahlen in Fließkommezahlen von Typ: LREAL nicht unterstützt. Vorzeichenlose Zahlen mit einem gesetzten höchstwertigen Bit werden möglicherweise implizit in negative Fließkommazahlen konvertiert. Die hier beschriebene Funktion erlaubt in TwinCAT 2 eine explizite Konvertierung vom Typ DWORD in eine positive Fließkommazahl vom Typ LREAL (auch wenn das höchstwertige Bit gesetzt wurde und ohne Compiler-Warnung). In TwinCAT 3 werden vorzeichnlose Zahlen von Typ DWORD immer (implizit und explizit) in eine positive Fließkommazahl konvertiert.

FUNCTION DWORD_TO_LREALEX : LREAL

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

in: Die zu konvertierende Zahl.

Beispiel:

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

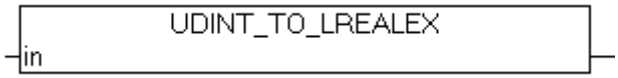
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nDword	-1, Warning 1105*	+4294967295	+4294967295
fLreal := DWORD_TO_LREAL(nDword)	-1, Warning 1105*	+4294967295	+4294967295
fLreal := DWORD#16#FFFFFFFF	-1, Warning 1105*	+4294967295	+4294967295
fLreal := 16#FFFFFFFF	+4294967295	+4294967295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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.72 UDINT_TO_LREALEX



In TwinCAT 2 auf ARM-Plattform wird die Konvertierung der vorzeichenlosen Zahlen in Fließkommezahlen von Typ: LREAL nicht unterstützt. Vorzeichenlose Zahlen mit einem gesetzten höchstwertigen Bit werden möglicherweise implizit in negative Fließkommazahlen konvertiert. Die hier beschriebene Funktion erlaubt in TwinCAT 2 eine explizite Konvertierung vom Typ UDINT in eine positive Fließkommazahl vom Typ LREAL (auch wenn das höchstwertige Bit gesetzt wurde und ohne Compiler-Warnung). In TwinCAT 3 werden vorzeichnlose Zahlen von Typ UDINT immer (implizit und explizit) in eine positive Fließkommazahl konvertiert.

FUNCTION UDINT_TO_LREALEX : LREAL

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

in: Die zu konvertierende Zahl.

Beispiel:

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

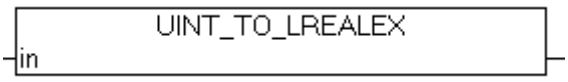
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nUdint	-1, Warning 1105*	+4294967295	+4294967295
fLreal := UDINT_TO_LREAL(nUdint)	-1, Warning 1105*	+4294967295	+4294967295
fLreal := UDINT#16#FFFFFFFF	-1, Warning 1105*	+4294967295	+4294967295
fLreal := 16#FFFFFFFF	+4294967295	+4294967295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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.73 UINT_TO_LREALEX



In TwinCAT 2 auf ARM-Plattform wird die Konvertierung der vorzeichenlosen Zahlen in Fließkommezahlen von Typ: LREAL nicht unterstützt. Vorzeichenlose Zahlen mit einem gesetzten höchstwertigen Bit werden möglicherweise implizit in negative Fließkommazahlen konvertiert. Die hier beschriebene Funktion erlaubt in TwinCAT 2 eine explizite Konvertierung vom Typ UINT in eine positive Fließkommazahl vom Typ LREAL (auch wenn das höchstwertige Bit gesetzt wurde und ohne Compiler-Warnung). In TwinCAT 3 werden vorzeichnlose Zahlen von Typ UINT immer (implizit und explizit) in eine positive Fließkommazahl konvertiert.

FUNCTION UINT_TO_LREALEX : LREAL

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

in: Die zu konvertierende Zahl.

Beispiel:

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

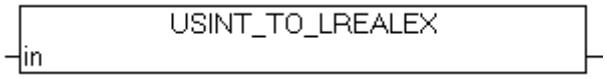
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := nUint	+65535, Warning 1105*	+65535	+65535
fLreal := UINT_TO_LREAL(nUint)	+65535, Warning 1105*	+65535	+65535
fLreal := UINT#16#FFFF	+65535, Warning 1105*	+65535	+65535
fLreal := 16#FFFF	+65535	+65535	+65535
fLreal := UINT_TO_LREALEX(nUint)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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.74 USINT_TO_LREALEX



In TwinCAT 2 auf ARM-Plattform wird die Konvertierung der vorzeichenlosen Zahlen in Fließkommezahlen von Typ: LREAL nicht unterstützt. Vorzeichenlose Zahlen mit einem gesetzten höchstwertigen Bit werden möglicherweise implizit in negative Fließkommazahlen konvertiert. Die hier beschriebene Funktion erlaubt in TwinCAT 2 eine explizite Konvertierung vom Typ USINT in eine positive Fließkommazahl vom Typ LREAL (auch wenn das höchstwertige Bit gesetzt wurde und ohne Compiler-Warnung). In TwinCAT 3 werden vorzeichnlose Zahlen von Typ USINT immer (implizit und explizit) in eine positive Fließkommazahl konvertiert.

FUNCTION USINT_TO_LREALEX : LREAL

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

in: Die zu konvertierende Zahl.

Beispiel:

```
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

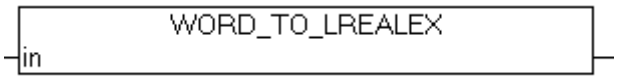
fLreal value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
fLreal := USINT_TO_LREAL(nUsint)	+255, Warning 1105*	+255	+255
fLreal := USINT#16#FF	+255, Warning 1105*	+255	+255
fLreal := 16#FF	+255	+255	+255
fLreal := USINT_TO_LREALEX(nUsint)	+255	+255	+255
fLreal := USINT_TO_LREALEX(USINT#16#FF)	+255	+255	+255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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.75 WORD_TO_LREALEX



In TwinCAT 2 auf ARM-Plattform wird die Konvertierung der vorzeichenlosen Zahlen in Fließkommezahlen von Typ: LREAL nicht unterstützt. Vorzeichenlose Zahlen mit einem gesetzten höchstwertigen Bit werden möglicherweise implizit in negative Fließkommazahlen konvertiert. Die hier beschriebene Funktion erlaubt in TwinCAT 2 eine explizite Konvertierung vom Typ WORD in eine positive Fließkommazahl vom Typ LREAL (auch wenn das höchstwertige Bit gesetzt wurde und ohne Compiler-Warnung). In TwinCAT 3 werden vorzeichnlose Zahlen von Typ WORD immer (implizit und explizit) in eine positive Fließkommazahl konvertiert.

FUNCTION WORD_TO_LREALEX : LREAL

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

in: Die zu konvertierende Zahl.

Beispiel:

```
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 value	Tc2.x ARM	Tc2.x X86	Tc3.x ARM, X86, X64
`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

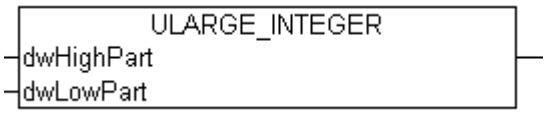
* Conversion of unsigned integer to LREAL is not supported. The value is used as signed instead.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2255	PC or CX (x86, ARM)	TcUtilities.Lib

3.76 Unsigned 64 bit integer

3.76.1 ULARGE_INTEGER



Die Funktion initialisiert eine vorzeichenlose 64 bit Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION ULARGE_INTEGER: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

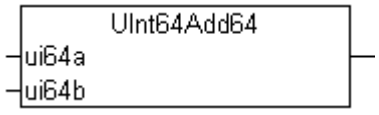
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.2 UInt64Add64



Die Funktion addiert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Add64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

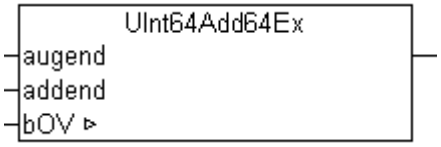
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.76.3 UInt64Add64Ex



Die Funktion addiert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Add64Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

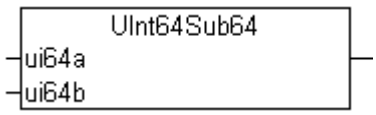
VAR_INPUT

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch eingebunden)
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcUtilities.Lib

3.76.4 UInt64Sub64



Die Funktion substrahiert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Sub64 : T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

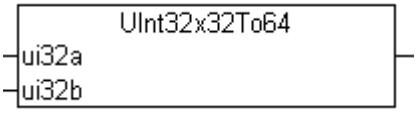
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.76.5 UInt32x32To64



Die Funktion multipliziert zwei vorzeichenlose 32 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt32x32To64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

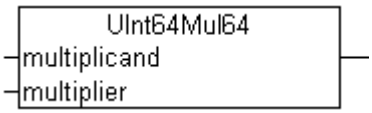
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.76.6 UInt64Mul64



Die Funktion multipliziert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Mul64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.7 UInt64Mul64Ex

Die Funktion multipliziert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Mul64Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

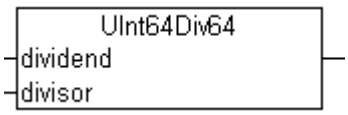
VAR_INPUT

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.8 UInt64Div64



Die Funktion dividiert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Div64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.9 UInt64Div64Ex

Die Funktion dividiert zwei vorzeichenlose 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Div64Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.10 UInt64Div16Ex

Die Funktion dividiert eine TwinCAT 2 vorzeichenlose 64 Bit Zahl („legacy“-Typ: T_ULARGE_INTEGER [▶ 240]) durch eine 16 Bit vorzeichenlose Zahl. Das Ergebnis ist eine vorzeichenlose 64 Bit Zahl.

FUNCTION UInt64Div16Ex: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

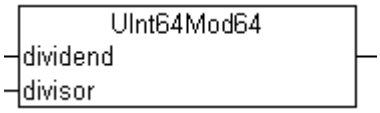
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >=1301	PC or CX (x86, ARM)	TcUtilities.Lib

3.76.11 UInt64Mod64



Modulo-Division einer vorzeichenlosen 64 bit Zahl durch eine andere Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Mod64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

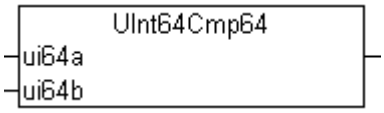
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.12 UInt64Cmp64



Die Funktion vergleicht zwei vorzeichenlose 64 bit Zahlen.

FUNCTION UInt64Cmp64: DINT

VAR_INPUT

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

Rückgabeparameter	Beschreibung
-1	ui64a kleiner als *ui64b*
0	ui64a identisch mit *ui64b*
1	ui64a größer als *ui64b*

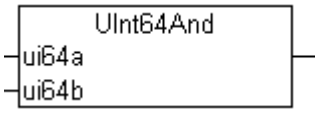
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

Sehen Sie dazu auch

- T_ULARGE_INTEGER [▶ 240]

3.76.13 UInt64And



Bitweise AND von zwei vorzeichenlosen 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64And: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

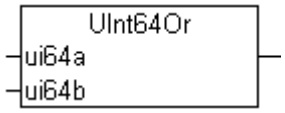
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.14 UInt64Or



Bitweise OR von zwei vorzeichenlosen 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Or: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

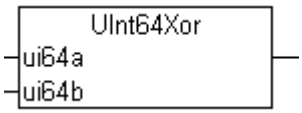
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.15 UInt64Xor



Bitweise XOR von zwei vorzeichenlosen 64 bit Zahlen. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Xor: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

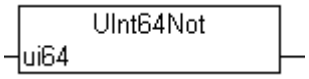
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.16 UInt64Not



Bitweise NOT einer vorzeichenlosen 64 bit Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Not: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

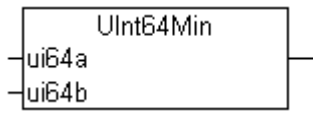
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.17 UInt64Min



Minimumsfunktion. Liefert von zwei Werten den kleineren Wert.

FUNCTION UInt64Min: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

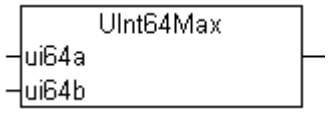
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.18 UInt64Max



Maximumsfunktion. Liefert von zwei Werten den größeren Wert.

FUNCTION UInt64Max: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

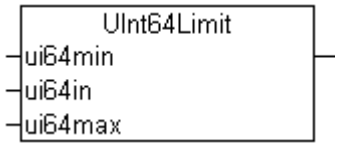
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.19 UInt64Limit



Limitierung. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Limit: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

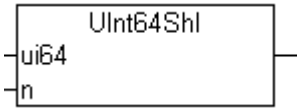
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.20 UInt64Shl



Bitweises Links-Shift einer vorzeichenlosen 64 bit Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Shl: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

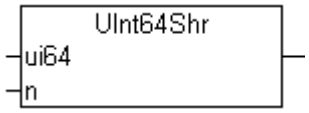
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.21 UInt64Shr



Bitweises Rechts-Shift einer vorzeichenlosen 64 bit Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Shr: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

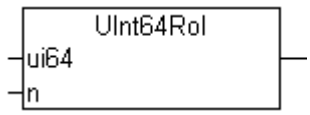
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.22 UInt64Rol



Bitweise Linksrotation einer vorzeichenlosen 64 bit Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Rol: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

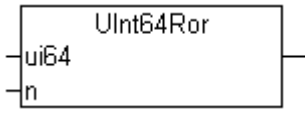
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.23 UInt64Ror



Bitweise Rechtsrotation einer vorzeichenlosen 64 bit Zahl. Das Ergebnis ist eine vorzeichenlose 64 bit Zahl.

FUNCTION UInt64Ror: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

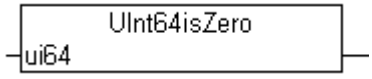
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.24 UInt64isZero



Die Funktion liefert TRUE wenn die vorzeichenlose 64 bit Zahl den Wert Null hat.

FUNCTION UInt64isZero: BOOL

VAR_INPUT

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

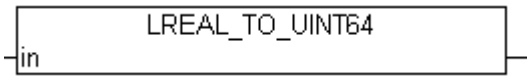
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

T_ULARGE_INTEGER [▶ 240]

3.76.25 LREAL_TO_UINT64



Die Funktion konvertiert eine LREAL Zahl in eine vorzeichenlose 64 bit Zahl.

FUNCTION LREAL_TO_UINT64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

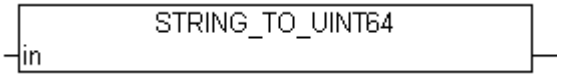
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.26 STRING_TO_UINT64



Die Funktion konvertiert einen String in eine vorzeichenlose 64 bit Zahl.

FUNCTION STRING_TO_UINT64: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

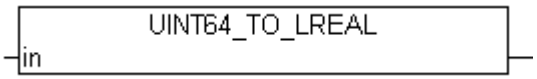
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.76.27 UINT64_TO_LREAL



Die Funktion konvertiert eine vorzeichenlose 64 bit Zahl in eine Fließkommazahl vom Typ LREAL.

FUNCTION UINT64_TO_LREAL: LREAL

VAR_INPUT

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

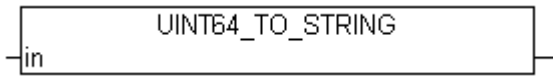
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

T_ULARGE_INTEGER [▶ 240]

3.76.28 UINT64_TO_STRING



Die Funktion konvertiert eine vorzeichenlose 64 bit Zahl in einen String.

FUNCTION UINT64_TO_STRING: STRING(21)

VAR_INPUT

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

Voraussetzungen

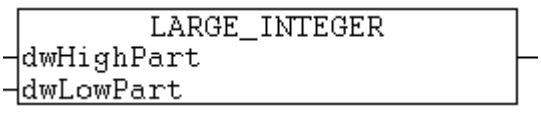
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1240	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

T_ULARGE_INTEGER [▶ 240]

3.77 Signed 64 bit integer

3.77.1 LARGE_INTEGER



Die Funktion initialisiert eine vorzeichenbehaftete 64 bit Zahl. Das Ergebnis ist eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION LARGE_INTEGER: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

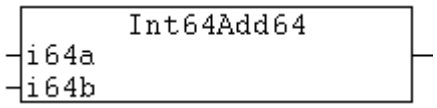
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.2 Int64Add64



Die Funktion addiert zwei vorzeichenbehaftete 64 bit Zahlen. Das Ergebnis ist eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION Int64Add64: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

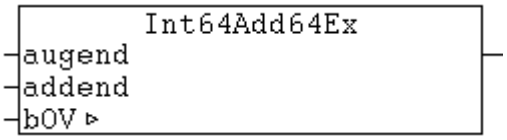
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.3 Int64Add64Ex



Die Funktion addiert zwei vorzeichenbehaftete 64 bit Zahlen. Das Ergebnis ist eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION Int64Add64Ex: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

VAR_INPUT

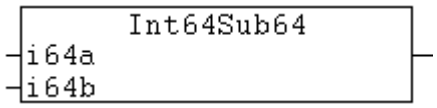
```
VAR_INPUT
    augend : T_LARGE_INTEGER;
    addend : T_LARGE_INTEGER;
END_VAR
```

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.4 Int64Sub64



Die Funktion substrahiert zwei vorzeichenbehaftete 64 bit Zahlen. Das Ergebnis ist eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION Int64Sub64 : T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

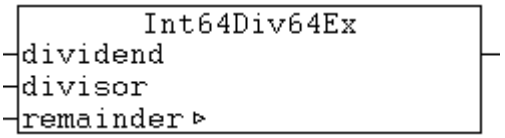
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.5 Int64Div64Ex



Die Funktion dividiert zwei vorzeichenbehaftete 64 bit Zahlen. Das Ergebnis ist eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION Int64Div64Ex: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

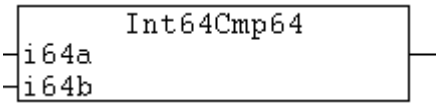
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.6 Int64Cmp64



Die Funktion vergleicht zwei vorzeichenbehaftete 64 bit Zahlen.

FUNCTION Int64Cmp64: DINT

VAR_INPUT

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

Rückgabeparameter	Beschreibung
-1	i64a kleiner als *i64b*
0	i64a identisch mit *i64b*
1	i64a größer als *i64b*

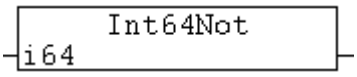
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_LARGE_INTEGER [▶ 240]

3.77.7 Int64Not



Bitweise NOT einer vorzeichenbehafteten 64 bit Zahl. Das Ergebnis ist eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION Int64Not: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

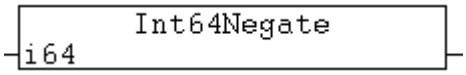
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.8 Int64Negate



Die Funktion negiert eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION Int64Negate: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

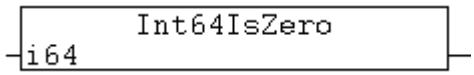
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.9 Int64isZero



Die Funktion liefert TRUE wenn die vorzeichenbehaftete 64 bit Zahl den Wert Null hat.

FUNCTION Int64isZero: BOOL

VAR_INPUT

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

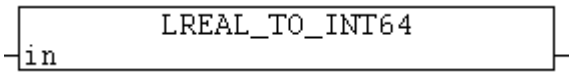
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_LARGE_INTEGER [▶ 240]

3.77.10 LREAL_TO_INT64



Die Funktion konvertiert eine LREAL Zahl in eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION LREAL_TO_INT64: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

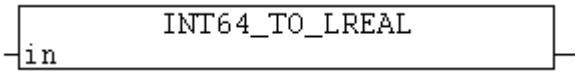
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.11 INT64_TO_LREAL



Die Funktion konvertiert eine vorzeichenbehaftete 64 bit Zahl in eine Fließkommazahl vom Typ LREAL.

FUNCTION INT64_TO_LREAL: LREAL

VAR_INPUT

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

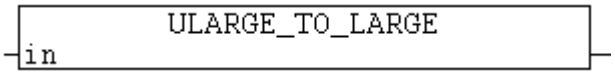
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_LARGE_INTEGER [▶ 240]

3.77.12 ULARGE_TO_LARGE



Die Funktion konvertiert eine vorzeichenlose 64 bit Zahl in eine vorzeichenbehaftete 64 bit Zahl.

FUNCTION ULARGE_TO_LARGE: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

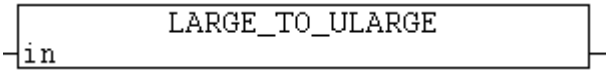
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1321	PC or CX (x86, ARM)	TcUtilities.Lib

3.77.13 LARGE_TO_ULARGE



Die Funktion konvertiert eine vorzeichenbehaftete 64 bit Zahl in eine vorzeichenlose 64 bit Zahl.

FUNCTION LARGE_TO_ULARGE: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

VAR_INPUT

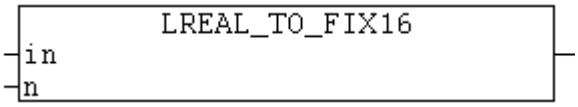
```
VAR_INPUT
    in : T_LARGE_INTEGER;
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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



Konvertiert eine Fließkommazahl vom Typ: LREAL in eine vorzeichenbehaftete 16 Bit Festkommazahl mit einer gewünschten Anzahl der Nachkommastellen.

FUNCTION LREAL_TO_FIX16 : T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

in: Die zu konvertierende LREAL-Zahl.

n: Anzahl der gewünschten Nachkommastellen.

Beispiel:

Im folgenden Beispiel werden einige Konstanten in Festkommazahlen konvertiert. Die Anzahl der Nachkommastellen kann bei der Konvertierung festgelegt werden. Bitte beachten Sie, dass bei der Konvertierung ähnlich wie bei den Fließkommazahlen Rundungsfehler entstehen können (in unserem Beispiel: q2 und q15).

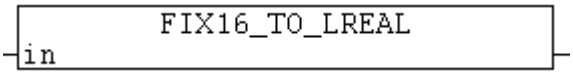
```
PROGRAM TEST
VAR
    q2, q4, q8, q12, q15 : T_FIX16;
    r2, r4, r8, r12, r15 : LREAL;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.2 FIX16_TO_LREAL



Konvertiert eine vorzeichenbehaftete 16 Bit Festkommazahl in eine Fließtkommazahl von Typ: LREAL.

FUNCTION FIX16_TO_LREAL : LREAL

VAR_INPUT

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

in: Die zu konvertierende Festkommazahl [▶ 241].

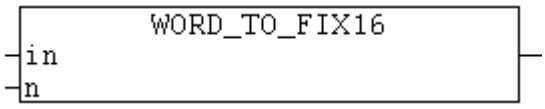
Beispiel:

Siehe in der Beschreibung der Funktion: LREAL_TO_FIX16 [▶ 97].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.3 WORD_TO_FIX16



Die Funktion konvertiert eine WORD-Variable in eine 16 Bit Festkommazahl (die WORD-Variable beinhaltet die kodierten Vor- und Nachkommastellen der Festkommazahl).

FUNCTION WORD_TO_FIX16 : T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

Beispiel:

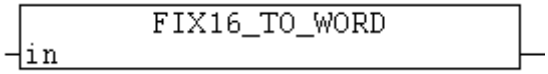
```
PROGRAM WORD_TO_FIX
VAR
    double: LREAL;
END_VAR
double := FIX16_TO_LREAL(WORD_TO_FIX16(2#0000110010000000, 8));
```

Der Wert der *double*-Variablen ist: 12.5

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.4 FIX16_TO_WORD



Diese Funktion konvertiert eine 16 Bit Festkommazahl in eine WORD-Variable (die WORD-Variable beinhaltet die Vor- und Nachkommastellen der Festkommazahl).

FUNCTION FIX16_TO_WORD : WORD

VAR_INPUT

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

Beispiel:

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

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

Der Wert der *fp16*-Variablen ist: 2#0000110010000000.

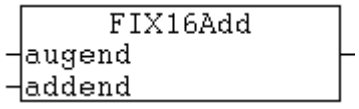
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_FIX16 [▶ 241]

3.78.5 FIX16Add



Diese Funktion addiert zwei vorzeichenbehaftete 16 Bit Festkommazahlen. Die Auflösung (Anzahl der Nachkommastellen) der Zahlen muss nicht gleich sein. Die Auflösung der Zahl mit der höheren Anzahl an Nachkommastellen wird vor der Addition reduziert. D.h. die Nachkommastellen der Zahl mit der höheren Auflösung werden abgeschnitten. Das Ergebnis der Addition ist eine vorzeichenbehaftete 16 Bit Festkommazahl.

FUNCTION FIX16Add : T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

augend: Der erste Summand [▶ 241].

addend: Der zweite Summand.

Beispiel:

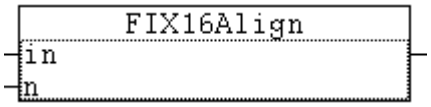
```
PROGRAM FIXADD
VAR
    a, b   : T_FIX16;
    result : LREAL;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.6 FIX16Align



Mit dieser Funktion kann die Auflösung (Anzahl der Nachkommastellen) einer vorzeichenbehafteten 16 Bit Festkommazahl geändert werden. Die neue Festkommazahl wird von der Funktion als Rückgabeparameter geliefert.

FUNCTION FIX16Align: T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

in: Festkommazahl [▶ 241] deren Auflösung geändert werden soll.

n: Die neue Anzahl der Nachkommastellen.

Beispiel:

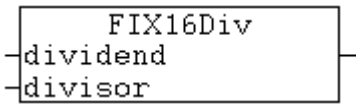
```
PROGRAM FIXALIGN
VAR
    q8, q4 : T_FIX16;
    result : LREAL;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.7 FIX16Div



Diese Funktion dividiert zwei vorzeichenbehaftete 16 Bit Festkommazahlen. Die Auflösung (Anzahl der Nachkommastellen) der Zahlen muss nicht gleich sein. Die Auflösung der Zahl mit der höheren Anzahl an Nachkommastellen wird vor der Division reduziert. D.h. die Nachkommastellen der Zahl mit der höheren Auflösung werden abgeschnitten. Das Ergebnis der Division ist eine vorzeichenbehaftete 16 Bit Festkommazahl.

FUNCTION FIX16Div: T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

dividend: Zahl [▶ 241], die geteilt wird.

divisor: Zahl, durch die geteilt wird.

Beispiel:

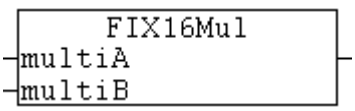
```
PROGRAM FIXDIV
VAR
    a, b : T_FIX16;
    result : LREAL;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.8 FIX16Mul



Diese Funktion multipliziert zwei vorzeichenbehaftete 16 Bit Festkommazahlen. Die Auflösung (Anzahl der Nachkommastellen) der Zahlen muss nicht gleich sein. Die Auflösung der Zahl mit der höheren Anzahl an Nachkommastellen wird vor der Multiplikation reduziert. D.h. die Nachkommastellen der Zahl mit der höheren Auflösung werden abgeschnitten. Das Ergebnis der Multiplikation ist eine vorzeichenbehaftete 16 Bit Festkommazahl.

FUNCTION FIX16Mul: T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

multiA: Der erste Multiplikator [▶ 241].

multiB: Der zweite Multiplikator.

Beispiel:

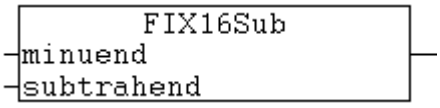
```
PROGRAM FIXMUL
VAR
    a, b : T_FIX16;
    result : LREAL;
END_VAR
```

```
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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.78.9 FIX16Sub



Diese Funktion subtrahiert zwei vorzeichenbehaftete 16 Bit Festkommazahlen. Die Auflösung (Anzahl der Nachkommastellen) der Zahlen muss nicht gleich sein. Die Auflösung der Zahl mit der höheren Anzahl an Nachkommastellen wird vor der Subtraktion reduziert. D.h. die Nachkommastellen der Zahl mit der höheren Auflösung werden abgeschnitten. Das Ergebnis der Subtraktion ist eine vorzeichenbehaftete 16 Bit Festkommazahl.

FUNCTION FIX16Sub : T_FIX16

T_FIX16 [▶ 241]

VAR_INPUT

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

minuend: Zahl [▶ 241] von der etwas abgezogen wird.

subtrahend: Zahl die abgezogen wird.

Beispiel:

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

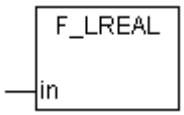
```
a := LREAL_TO_FIX16( 0.5, 8 );
b := LREAL_TO_FIX16( 0.75, 8 );

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.79 Hilfsfunktionen

3.79.1 F_LREAL



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer LREAL-Variablen zurückliefert.

FUNCTION F_LREAL : T_Arg

T_Arg [▶ 239]

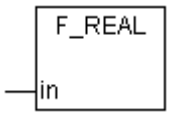
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.2 F_REAL



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer REAL-Variablen zurückliefert.

FUNCTION F_REAL : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.3 F_BYTE

Eine Hilfsfunktion, die in einer Struktur Informationen zu einer BYTE-Variablen zurückliefert.

FUNCTION F_BYTE : T_Arg

T_Arg [▶ 239]

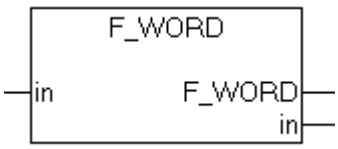
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.4 F_WORD



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer WORD-Variablen zurückliefert.

FUNCTION F_WORD : T_Arg

T_Arg [▶ 239]

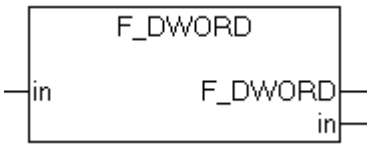
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.5 F_DWORD



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer DWORD-Variablen zurückliefert.

FUNCTION F_DWORD : T_Arg

T_Arg [▶ 239]

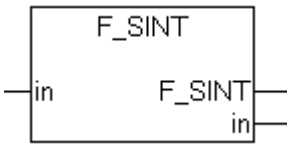
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.6 F_SINT



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer SINT-Variablen zurückliefert.

FUNCTION F_SINT : T_Arg

T_Arg [▶ 239]

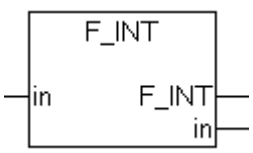
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.7 F_INT



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer INT-Variablen zurückliefert.

FUNCTION F_INT : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.8 F_DINT

Eine Hilfsfunktion, die in einer Struktur Informationen zu einer DINT-Variablen zurückliefert.

FUNCTION F_DINT : T_Arg

T_Arg [▶ 239]

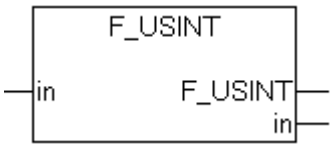
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.9 F_USINT



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer USINT-Variablen zurückliefert.

FUNCTION F_USINT : T_Arg

T_Arg [▶ 239]

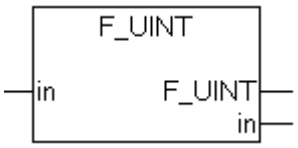
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.10 F_UINT



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer UINT-Variablen zurückliefert.

FUNCTION F_UINT : T_Arg

T_Arg [▶ 239]

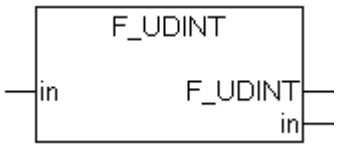
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.11 F_UDINT



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer UDINT-Variablen zurückliefert.

FUNCTION F_UDINT : T_Arg

T_Arg [▶ 239]

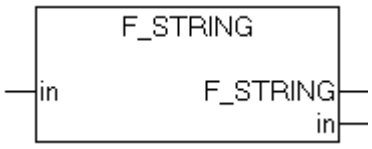
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.12 F_STRING



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer T_MaxString-Variablen zurückliefert.

FUNCTION F_STRING : T_Arg

T_Arg [▶ 239]

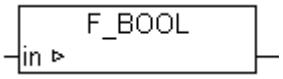
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.13 F_BOOL



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer BOOL-Variablen zurückliefert.

FUNCTION F_BOOL : T_Arg

T_Arg [▶ 239]

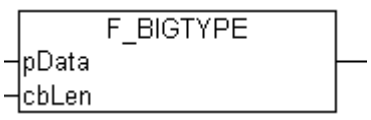
VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

3.79.14 F_BIGTYPE



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer Struct- oder Array-Variablen zurückliefert.

FUNCTION F_BIGTYPE : T_Arg

T_Arg [▶ 239]

VAR_INPUT

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

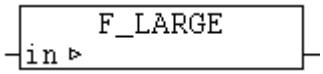
pData: Addresspointer (kann mit dem ADR-Operator ermittelt werden).

cbLen: Anzahl der Bytes, die im Speicher belegt werden (kann mit SIZEOF-Operator ermittelt werden).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1235	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

3.79.15 F_LARGE



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer T_LARGE_INTEGER-Variablen zurückliefert (signed 64 bit integer).

FUNCTION F_LARGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

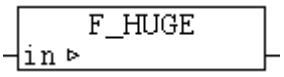
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_LARGE_INTEGER [▶ 240]

3.79.16 F_HUGE



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer T_HUGE_INTEGER-Variablen zurückliefert (signed 128 bit integer).

FUNCTION F_HUGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

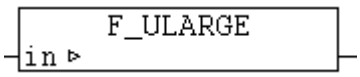
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_HUGE_INTEGER [▶ 241]

3.79.17 F_ULARGE



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer T_ULARGE_INTEGER-Variablen zurückliefert (unsigned 64 bit number).

FUNCTION F_ULARGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

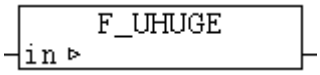
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_ULARGE_INTEGER [▶ 240]

3.79.18 F_UHUGE



Eine Hilfsfunktion, die in einer Struktur Informationen zu einer T_UHUGE_INTEGER-Variablen zurückliefert (unsigned 128 bit integer).

FUNCTION F_UHUGE : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

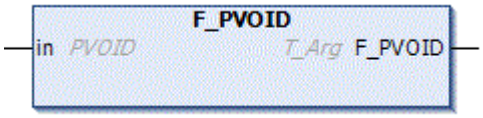
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_UHUGE_INTEGER [▶ 240]

3.79.19 F_PVOID



Eine Hilfsfunktion, die in einer Struktur Informationen (Typ: T_Arg [▶ 239]) zu einer PVOID-Variablen zurückliefert.

FUNCTION F_PVOID : T_Arg

T_Arg [▶ 239]

VAR_IN_OUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

3.79.20 F_ARGCPY

Diese Funktion kopiert den Wert einer Variablen vom Typ T_Arg in eine andere Variable und liefert die Anzahl der erfolgreich kopierten Datenbytes als Rückgabeparameter zurück.

FUNCTION F_ARGCPY: UDINT

VAR_INPUT

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

typeSafe: Wenn TRUE => Gleiche Typen können verglichen werden (Typsicherer-Vergleich). FALSE => Unterschiedliche Typen können verglichen werden (Typunabhängiger-Vergleich).

VAR_IN_OUT

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

dest: Zielvariable [▶ 239] in die hineinkopiert werden soll.

src: Quellvariable von der kopiert werden soll.

Rückgabeparameter	Bedeutung
0	Falsche Parameterwerte. Der Typ, Länge oder Wert von *dest* oder *src* == 0
> 0	Bei Erfolg, die Anzahl der kopierten Bytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.79.21 F_ARGCMP

Diese Funktion vergleicht zwei Variablen vom Typ T_Arg und liefert das Ergebnis des Vergleichs als Rückgabeparameter zurück.

FUNCTION F_ARGCMP: DINT

VAR_INPUT

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

typeSafe: Wenn TRUE => Gleiche Typen können verglichen werden (Typsicherer-Vergleich). FALSE => Unterschiedliche Typen können verglichen werden (Typunabhängiger-Vergleich).

arg1: Erste Variable [▶ 239] die verglichen werden soll.

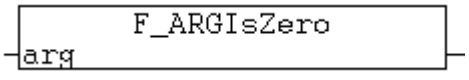
arg2: Zweite Variable die verglichen werden soll.

Rückgabeparameter	Verhältnis des ersten unterschiedlichen Bytes (Typ, Länge, Wert) in der ersten und zweiten Variablen
-3	Länge von *arg1* kleiner als *arg2*
-2	Typ von *arg1* kleiner als *arg2*
-1	Wert von *arg*1 kleiner als *arg2*
0	arg1 identisch mit *arg2*
1	Wert *von arg1* größer als *arg2*
2	Typ von *arg1* größer als *arg2*
3	Länge von *arg1* größer als *arg2*
0xFF	Falsche Parameterwerte, Typ, Länge, Wert von *arg1 oder arg2* = 0

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (x86, ARM)	TcUtilities.Lib

3.79.22 F_ARGISZERO



Diese Funktion liefert TRUE wenn eine der T_Arg-Membervariablen den Wert Null hat oder nicht initialisiert wurde.

FUNCTION F_ARGISZERO: BOOL

VAR_INPUT

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

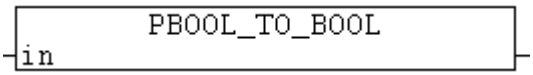
arg: Variable [▶ 239] die geprüft werden soll.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.80 P[TYPE]_TO_[TYPE]-Konvertierung

3.80.1 PBOOL_TO_BOOL



Die Funktion liefert den Inhalt einer BOOL-Pointervariablen.

FUNCTION PBOOL_TO_BOOL: BOOL

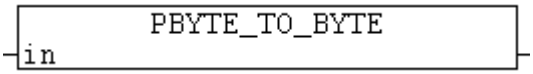
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.2 PBYTE_TO_BYTE



Die Funktion liefert den Inhalt einer BYTE-Pointervariablen.

FUNCTION PBYTE_TO_BYTE: BYTE

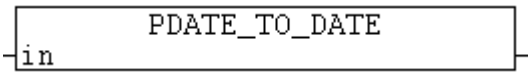
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.3 PDATE_TO_DATE



Die Funktion liefert den Inhalt einer DATE-Pointervariablen.

FUNCTION PDATE_TO_DATE: DATE

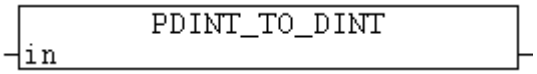
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.4 PDINT_TO_DINT



Die Funktion liefert den Inhalt einer DINT-Pointervariablen.

FUNCTION PDINT_TO_DINT: DINT

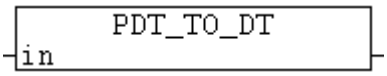
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.5 PDT_TO_DT



Die Funktion liefert den Inhalt einer DT-Pointervariablen.

FUNCTION PDT_TO_DT: DT

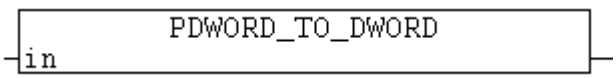
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.6 PDWORD_TO_DWORD



Die Funktion liefert den Inhalt einer DWORD-Pointervariablen.

FUNCTION PDWORD_TO_DWORD: DWORD

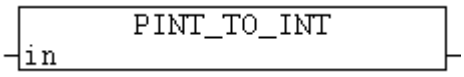
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.7 PINT_TO_INT



Die Funktion liefert den Inhalt einer INT-Pointervariablen.

FUNCTION PINT_TO_INT: INT

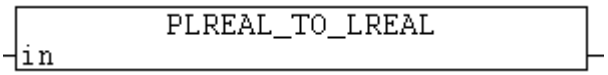
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.8 PLREAL_TO_LREAL



Die Funktion liefert den Inhalt einer LREAL-Pointervariablen.

FUNCTION PLREAL_TO_LREAL: LREAL

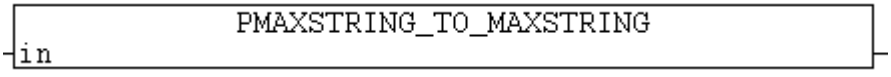
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.9 PMAXSTRING_TO_MAXSTRING



Die Funktion liefert den Inhalt einer T_MaxString-Pointervariablen.

FUNCTION PMAXSTRING_TO_MAXSTRING: T_MaxString

T_MaxString

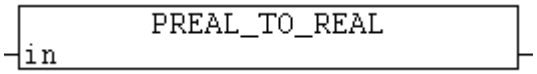
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.10 PREAL_TO_REAL



Die Funktion liefert den Inhalt einer REAL-Pointervariablen.

FUNCTION PREAL_TO_REAL: REAL

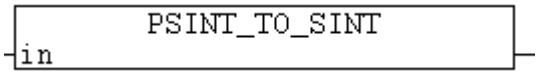
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.11 PSINT_TO_SINT



Die Funktion liefert den Inhalt einer SINT-Pointervariablen.

FUNCTION PSINT_TO_SINT: SINT

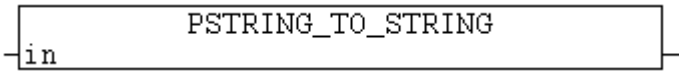
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.12 PSTRING_TO_STRING



Die Funktion liefert den Inhalt einer STRING-Pointervariablen.

FUNCTION PSTRING_TO_STRING: STRING

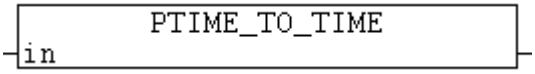
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.13 PTIME_TO_TIME



Die Funktion liefert den Inhalt einer TIME-Pointervariablen.

FUNCTION PTIME_TO_TIME: TIME

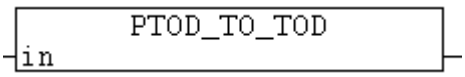
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.14 PTOD_TO_TOD



Die Funktion liefert den Inhalt einer TOD-Pointervariablen.

FUNCTION PTOD_TO_TOD: TOD

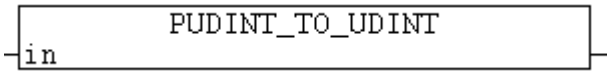
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.15 PUDINT_TO_UDINT



Die Funktion liefert den Inhalt einer UDINT-Pointervariablen.

FUNCTION PUDINT_TO_UDINT: UDINT

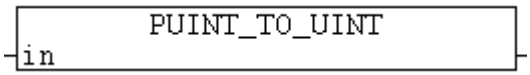
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.16 PUINT_TO_UINT



Die Funktion liefert den Inhalt einer UINT-Pointervariablen.

FUNCTION PUINT_TO_UINT: UINT

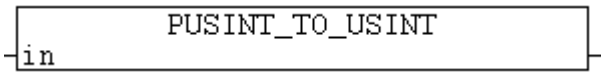
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.17 PUSINT_TO_USINT



Die Funktion liefert den Inhalt einer USINT-Pointervariablen.

FUNCTION PUSINT_TO_USINT: USINT

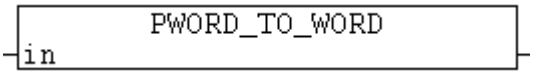
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.18 PWORD_TO_WORD



Die Funktion liefert den Inhalt einer WORD-Pointervariablen.

FUNCTION PWORD_TO_WORD: WORD

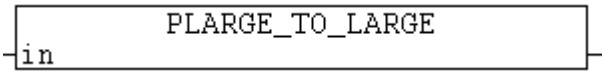
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1324	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.19 PLARGE_TO_LARGE



Die Funktion liefert den Inhalt einer T_LARGE_INTEGER-Pointervariablen.

FUNCTION PLARGE_TO_LARGE: T_LARGE_INTEGER

T_LARGE_INTEGER [▶ 240]

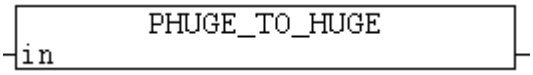
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.20 PHUGE_TO_HUGE



Die Funktion liefert den Inhalt einer T_HUGE_INTEGER-Pointervariablen.

FUNCTION PHUGE_TO_HUGE: T_HUGE_INTEGER

T_HUGE_INTEGER [▶ 241]

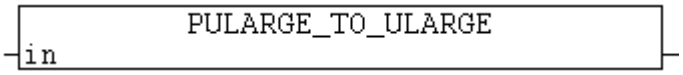
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.21 PULARGE_TO_ULARGE



Die Funktion liefert den Inhalt einer T_ULARGE_INTEGER-Pointervariablen.

FUNCTION PULARGE_TO_ULARGE: T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

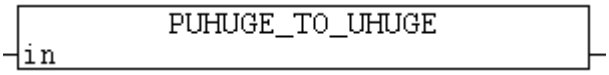
VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.80.22 PUHUGE_TO_UHUGE



Die Funktion liefert den Inhalt einer T_UHUGE_INTEGER-Pointervariablen.

FUNCTION PUHUGE_TO_UHUGE: T_UHUGE_INTEGER

T_UHUGE_INTEGER [▶ 240]

VAR_INPUT

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

3.81 Byte Order-Konvertierung

3.81.1 Host Byte Order / Network Byte Order

Bei Netzwerkprotokollen wird die Byte-Reihenfolge festgeschrieben. Diese wird als Netzwerk-Byte-Reihenfolge (Network Byte Order) bezeichnet. Die natürliche Byte-Reihenfolge des TwinCAT-Systems wird als Host-Byte-Reihenfolge (Host Byte Order) bezeichnet. In den meisten Fällen entspricht die geforderte Netzwerk-Byte-Reihenfolge dem Big-Endian-Format (MOTOROLA). Das TwinCAT-SPS-System arbeitet aber mit dem Little-Endian-Format (INTEL). Damit ein fehlerfreier Datenaustausch zwischen dem TwinCAT-SPS-System und einer anderen Plattform stattfinden kann muss die Byte-Reihenfolge im Anwendungsprogramm entsprechend umgewandelt werden.

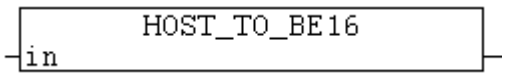
Daten, die über ein Netzwerkprotokoll vom TwinCAT System (Host) zu einem Fremdsystem übertragen werden sollen, können mit folgenden Funktionen ins Netzwerk-Format konvertiert werden:

- HOST_TO_BE16 [▶ 121]
- HOST_TO_BE32 [▶ 122]
- HOST_TO_BE64 [▶ 122]
- HOST_TO_BE128 [▶ 122]

Die empfangenen Netzwerkdaten (Fremdsystem) können wiederum mit folgenden Funktionen ins Host-Format (TwinCAT-System) konvertiert werden:

- BE16_TO_HOST [▶ 123]
- BE32_TO_HOST [▶ 123]
- BE64_TO_HOST [▶ 124]
- BE128_TO_HOST [▶ 124]

3.81.2 HOST_TO_BE16



Die Funktion führt eine Host-To-Network-Umwandlung einer 16 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION HOST_TO_BE16 : WORD

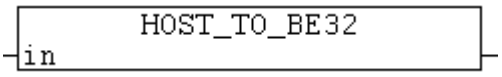
```
VAR_INPUT
    in      : WORD;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.3 HOST_TO_BE32



Die Funktion führt eine Host-To-Network-Umwandlung einer 32 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION HOST_TO_BE32 : DWORD

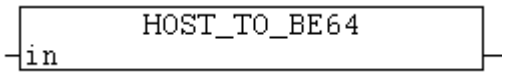
```
VAR_INPUT
    in      : DWORD;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.4 HOST_TO_BE64



Die Funktion führt eine Host-To-Network-Umwandlung einer 64 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION HOST_TO_BE64 : T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

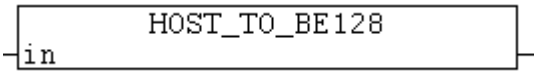
```
VAR_INPUT
    in      : T_ULARGE_INTEGER;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.5 HOST_TO_BE128



Die Funktion führt eine Host-To-Network-Umwandlung einer 128 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION HOST_TO_BE128 : T_UHUGE_INTEGER

T_UHUGE_INTEGER [▶ 240]

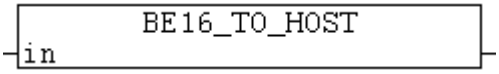
```
VAR_INPUT
    in      : T_UHUGE_INTEGER;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.6 BE16_TO_HOST



Die Funktion führt eine Network-To-Host-Umwandlung einer 16 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION BE16_TO_HOST : WORD

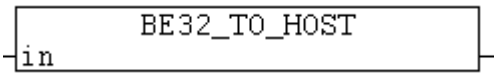
```
VAR_INPUT
    in      : WORD;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.7 BE32_TO_HOST



Die Funktion führt eine Network-To-Host-Umwandlung einer 32 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION BE32_TO_HOST : DWORD

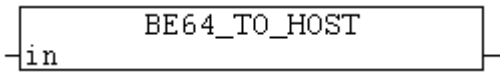
```
VAR_INPUT
    in      : DWORD;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.8 BE64_TO_HOST



Die Funktion führt eine Network-To-Host-Umwandlung einer 64 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION BE64_TO_HOST : T_ULARGE_INTEGER

T_ULARGE_INTEGER [▶ 240]

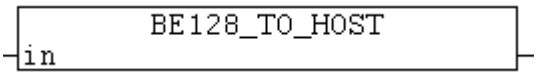
```
VAR_INPUT
    in      : T_ULARGE_INTEGER;
END_VAR
```

in: Die zu konvertierende Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

3.81.9 BE128_TO_HOST



Die Funktion führt eine Network-To-Host-Umwandlung einer 128 bit Zahl durch. Siehe auch unter: Byte Order [▶ 121].

FUNCTION BE128_TO_HOST : T_UHUGE_INTEGER

T_UHUGE_INTEGER [▶ 240]

```
VAR_INPUT
    in      : T_UHUGE_INTEGER;
END_VAR
```

in: Die zu konvertierende Zahl.

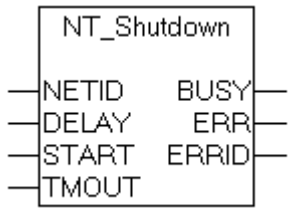
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

4 Funktionsbausteine

4.1 NT_Shutdown

Diese Funktionalität ist unter Windows CE nicht verfügbar!



Mit dem Funktionsbaustein "NT_Shutdown" kann ein Shutdown des Windows NT Betriebssystems durchgeführt werden. Die Funktion entspricht in etwa dem Shutdown-Befehl in der Windows-Startleiste. Über den Parameter DELAY kann eine Verzögerungszeit für die Ausführung des Shutdown-Befehls angegeben werden. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

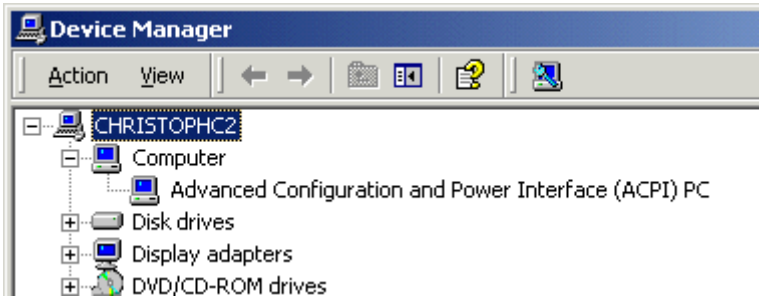
Bemerkungen:

Bei Betriebssystemen (ab Windows 2000) wird beim Aufruf des NT_Shutdown-Funktionsbausteins ein Shutdown mit Power OFF ausgeführt. D.h. der PC schaltet sich nach dem Shutdown Vorgang selbsttätig ab. Diese Funktion steht aber nur dann zur Verfügung wenn von dem Betriebssystem ein ACPI-fähiger PC erkannt wurde (Advanced Configuration and Power Interface). Die ACPI-Funktionen müssen auch von dem Motherboard und dem Netzteil des PCs unterstützt werden. In den meisten Fällen muss die ACPI-Funktionsfähigkeit im BIOS vor der Installation des Betriebssystems aktiviert werden. Eine nachträgliche Änderung wird in den meisten Fällen von dem Betriebssystem nicht erkannt. Ob ein ACPI-fähiger PC erkannt wurde, kann z.B. bei Windows 2000 auf folgende Weise geprüft werden:

1. In der "Systemsteuerung" den Ordner "System" öffnen.

2. Auf dem Karteireiter "Hardware" den "Geräte Manager" auswählen.

In dem Baum mit den Geräten muss unter Computer folgendes stehen: "Advanced Configuration and Power Interface (ACPI) PC".



Beim Aufruf von NT_SHUTDOWN führt TwinCAT standardmäßig ein Shutdown mit POWER-OFF durch. Unterstützt ein PC diese Funktionalität nicht, dann kann diese durch einen Eintrag in der Windows-Registry deaktiviert werden. Folgender Wert muss in die Registry eingetragen werden (ab TwinCAT Version >= 2.7 Build >= 505):

"DisableACPIPowerOff" REG_DWORD = 0x00000001 in Registry unter:
"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: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem ein Shutdown des Betriebssystems durchgeführt werden soll. Für einen Shutdown auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

DELAY: Verzögerungszeit für die Ausführung des Shutdown-Befehls in Sekunden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

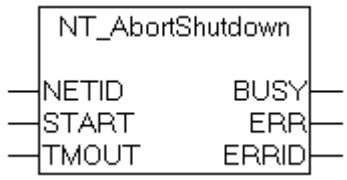
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

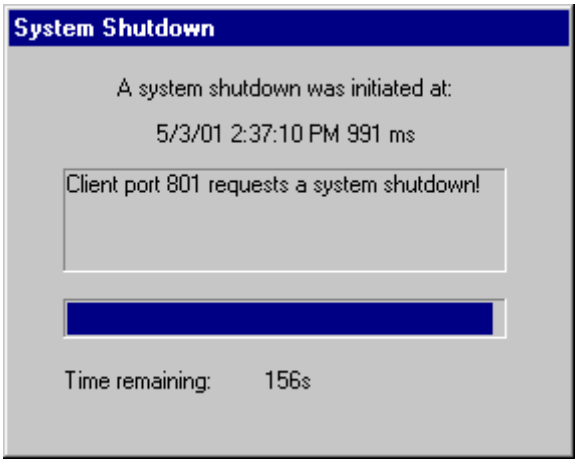
4.2 NT_AbortShutdown



Mit dem Funktionsbaustein "NT_AbortShutdown" kann ein zuvor mit dem NT_Shutdown [▶ 125]-Funktionsbaustein aufgerufenes Shutdown-Kommando abgebrochen werden. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

Bemerkung:

Beim Aufruf des NT_Shutdown [▶ 125]-Funktionsbausteines kann eine Verzögerungszeit als Parameter angegeben werden. Die verbleibende Zeit wird in einem Meldungsfenster angezeigt:



Erst nachdem die Verzögerungszeit abgelaufen ist, wird ein Shutdown des Betriebssystems durchgeführt. In dieser Zeit kann der Shutdown-Prozess mit dem Funktionsbaustein "NT_AbortShutdown" aus der SPS unterbrochen werden.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    START       : BOOL;
    TMOUT       : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem der Shutdown-Prozess abgebrochen werden soll soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

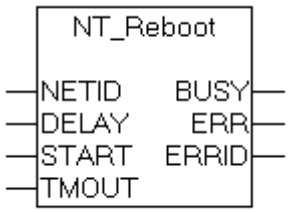
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.3 NT_Reboot



Mit dem Funktionsbaustein "NT_Reboot" kann ein Restart des Windows NT Betriebssystems durchgeführt werden. Die Funktion entspricht in etwa dem Restart-Befehl in der Windows-Startleiste. Über den Parameter DELAY kann eine Verzögerungszeit für die Ausführung des Restart-Befehls angegeben werden. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    DELAY       :DWORD;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen Restart durchgeführt werden soll. Für einen Restart des lokalen Rechners, kann auch ein Leerstring angegeben werden.

DELAY: Verzögerungszeit für die Ausführung des Restart-Befehls in Sekunden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

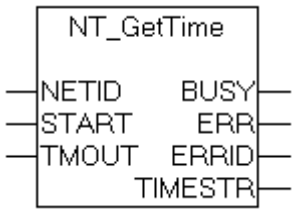
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.4 NT_GetTime



Mit dem Funktionsbaustein "NT_GetTime" kann die lokale Windows-Systemzeit eines TwinCAT-Systems ermittelt werden (die lokale Windows-Systemzeit wird in der Taskleiste eingeblendet). Das Jahr, Monat, Tag, Wochentag, Stunde, Minute, Sekunde und Millisekunde werden in den Variablen der Struktur TIMESTRUCT [▶ 231] abgelegt. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen lokale Windows-Systemzeit ermittelt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

TIMESTR: Struktur [▶ 231] mit der lokalen Windows-Systemzeit.

Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

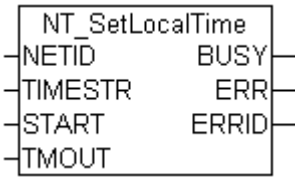
FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime, NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.5 NT_SetLocalTime



Mit dem Funktionsbaustein "NT_SetLocalTime" kann die lokale Windows-Systemzeit eines TwinCAT-Systems gesetzt werden (die lokale Windows-Systemzeit wird in der Taskleiste eingeblendet).

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TIMESTR     :TIMESTRUCT;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen lokale Windows-Systemzeit gesetzt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TIMESTR: Struktur [▶ 231] mit der neuen lokalen Windows-Systemzeit.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

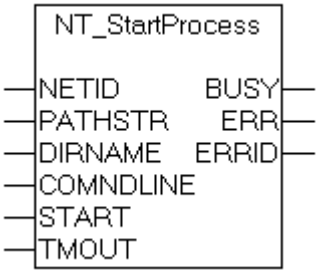
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime, NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1030 und höher TwinCAT v2.10.0 Build > 1232 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.6 NT_StartProcess



Mit dem Funktionsbaustein "NT_StartProcess" kann aus der SPS heraus eine Windows-Anwendung gestartet werden. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen. Mit dem Funktionsbaustein können auch Anwendungen auf einem Remote-PC gestartet werden.

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: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem die Anwendung gestartet werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

PATHSTR: Der gesamte Pfad der auszuführenden Anwendung als String (z.B. "C:\WINNT\NOTEPAD.EXE").

DIRNAME: Arbeitsverzeichnis der auszuführenden Anwendung als String (z.B. "C:\WINNT").

COMNDLINE: Kommandozeilen-Parameter (z.B.: "win.ini").

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

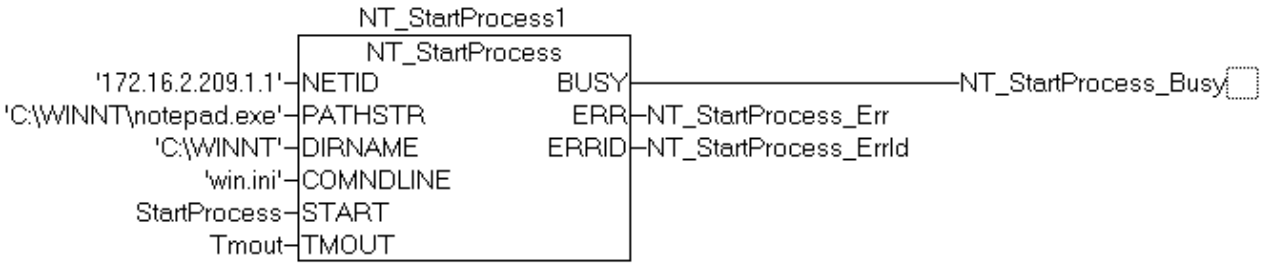
BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer oder den Win32-Fehlercode (Platform SDK: Win32 API).

Beispiel für einen Aufruf in FUP

```
NT_StartProcess1         : NT_StartProcess;
NT_StartProcess_Busy     : BOOL;
NT_StartProcess_Err      : BOOL;
NT_StartProcess_ErrId    : UDINT;
StartProcess             : BOOL;
Tmout                    : TIME;
```

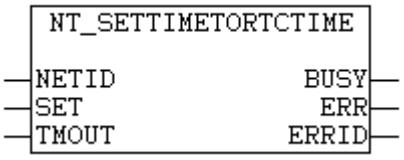


Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.7 NT_SetTimeToRTCTime

Diese Funktionalität ist in dem SPS-Laufzeitsystem unter Windows CE nicht verfügbar!



Mit dem Funktionsbaustein "NT_SetTimeToRTCTime" kann die lokale Windows-Systemzeit (wird in der Taskleiste eingeblendet) mit der Echtzeituhr des PCs (RTC-Zeit in BIOS) synchronisiert werden.

Bemerkungen

Beim Aufruf des Funktionsbausteins wird die Echtzeituhr des TwinCAT PCs mit der lokalen Windows-Systemzeit verglichen und die lokale Windows-Systemzeit um die ermittelte Differenz korrigiert. Zeitzonen und Sommerzeit werden dabei berücksichtigt. Zu beachten ist, dass es durch die Korrektur zu Zeitsprüngen während Messungen oder Logbuchaufzeichnungen führen kann.

Beim Setzen der lokalen Windows-Systemzeit wird vom Betriebssystem auch automatisch die RTC-Zeit auf die neue lokale Windows-Systemzeit gesetzt. Die neue RTC-Zeit wird dabei zwangsläufig durch die Umrechnung und Verzögerung mit einem kleinen Fehler behaftet. Der Fehler liegt im Millisekundenbereich.

D.h. mit jedem Aufruf von NT_SetTimeToRTCTime wird die Echtzeituhr ein wenig verfälscht. Um möglichst kleine Abweichungen über einen längeren Zeitraum zu erreichen, sollte der Abgleich z.B. alle 24 Stunden und nicht in jedem SPS-Zyklus durchgeführt werden.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    SET         :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem die lokale Windows-Systemzeit synchronisiert werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

SET: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

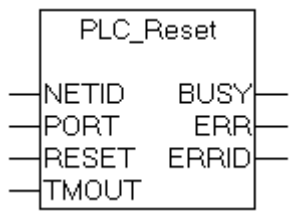
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime, F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
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



Mit dem Funktionsbaustein "PLC_Reset" kann ein Reset eines SPS-Laufzeitsystems durchgeführt werden. Bei einem Reset der SPS werden die SPS-Variablen mit den Initialwerten belegt und die Ausführung des SPS-Programms gestoppt. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

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

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem ein Reset eines SPS-Laufzeitsystems durchgeführt werden soll. Für einen SPS-Reset auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

PORT: Beinhaltet die ADS-Portnummer des SPS-Laufzeitsystems dessen Reset durchgeführt werden soll. Das Erste SPS-Laufzeitsystem hat z.B. die Portnummer 801 und das Zweite die Portnummer 811.

RESET: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

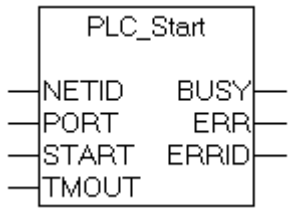
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.9 PLC_Start



Mit dem Funktionsbaustein "PLC_Start" kann ein SPS-Laufzeitsystem eines TwinCAT-Rechners gestartet werden. Der Funktionsbaustein kann z.B. dazu benutzt werden, um die SPS auf einem Remote-PC zu starten.

Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

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

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem die SPS gestartet werden soll. Für einen SPS-Start auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

PORT: Beinhaltet die ADS-Portnummer des SPS-Laufzeitsystems das gestartet werden soll. Das Erste SPS-Laufzeitsystem hat z.B. die Portnummer 801 und das Zweite die Portnummer 811.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

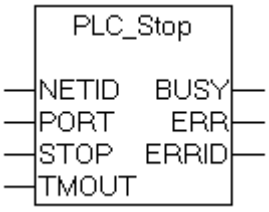
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.10 PLC_Stop



Mit dem Funktionsbaustein "PLC_Stop" kann ein SPS-Laufzeitsystem eines TwinCAT-Rechners gestoppt werden. Der Funktionsbaustein kann z.B. dazu benutzt werden, um die SPS auf einem Remote/Lokalen-PC zu stoppen. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    PORT        :T_AmsPort;
```

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

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem die SPS gestoppt werden soll. Für einen SPS-Stop auf dem lokalen Rechner, kann auch ein Leerstring angegeben werden.

PORT: Beinhaltet die ADS-Portnummer des SPS-Laufzeitsystems das gestoppt werden soll. Das Erste SPS-Laufzeitsystem hat z.B. die Portnummer 801 und das Zweite die Portnummer 811.

STOP: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

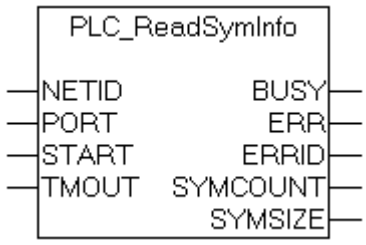
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.11 PLC_ReadSymInfo



Mit dem Funktionsbaustein "PLC_ReadSymInfo" können Informationen über die Symbole (Variablen) eines SPS-Laufzeitsystems ermittelt werden. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen.

VAR_INPUT

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

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen Symbolinformationen ermittelt werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

PORT: Die Portnummer eines SPS-Laufzeitsystems.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

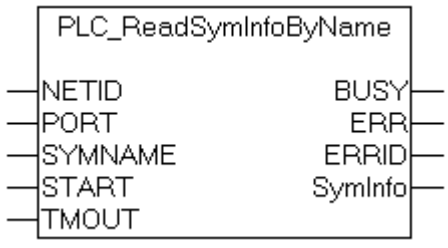
SYMCOUNT: Anzahl der Symbole in dem SPS-Laufzeitsystem.

SYMSIZE: Länge der Daten in Byte, in denen die Symbolinformationen gespeichert werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.12 PLC_ReadSymInfoByName



Mit dem Funktionsbaustein "PLC_ReadSymInfoByName" können zusätzliche Informationen über eine SPS-Symbolvariable (z.B. Datentypbezeichnung, Index-Group, Index-Offset, Kommentar...) anhand des Symbolnamens ausgelesen werden. Nach einer erfolgreichen Ausführung liegen die Daten in der Datenstruktur **SymInfo** vom Typ: SYMINFOSTRUCT [▶ 233] zur Verfügung. Die maximale Länge von dem Symbolnamen, der an den Funktionsbaustein als Parameter übergeben wird ist auf 255 Zeichen begrenzt.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    PORT        :T_AmsPort;
    SYMNAME     :T_MaxString;
```

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

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem Funktion ausgeführt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

PORT: Die Portnummer des SPS-Laufzeitsystems zu dem die Symbolvariable gehört. Das erste SPS-Laufzeitsystem besitzt die Portnummer 801.

SYMNAME: Der Symbolname der SPS-Variablen deren Informationen gelesen werden sollen (max. 255 Zeichen, inklusive des gesamten Pfades z.B. 'MAIN.INIT_TASK.VARINT').

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

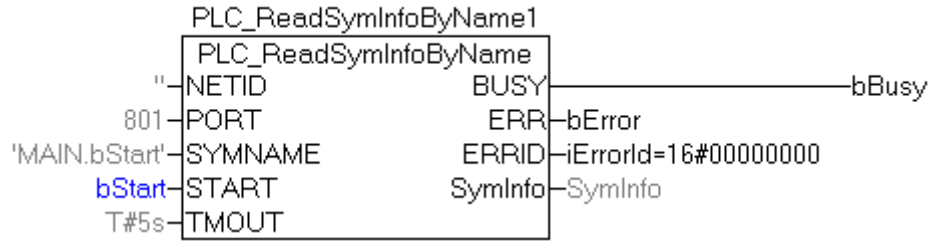
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

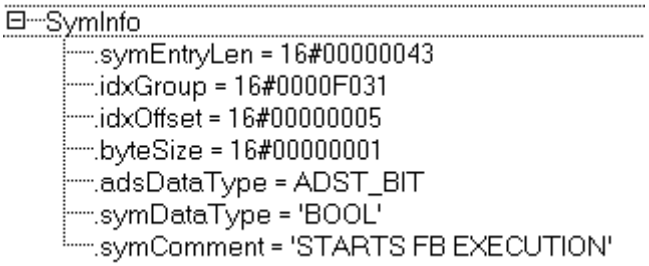
SymInfo: Struktur [▶ 233] mit zusätzlichen Informationen zu der Symbolvariablen.

Beispiel für einen Aufruf in FUP

```
PROGRAM MAIN
VAR
    PLC_ReadSymInfoByName1      : PLC_ReadSymInfoByName;
    bStart  AT%QX0.5            : BOOL;     (*Starts FB execution*)

    bBusy               : BOOL;
    bError              : BOOL;
    iErrorId            : UDINT;
    SymInfo             : SYMINFOSTRUCT;(*Structure with sombol information*)
END_VAR
```



Online-Ansicht:



Die erhaltenen Daten haben folgenden Bedeutung:

symEntryLen = 16#43: Der Eintrag in der Symboltabelle hat eine tatsächliche Größe von 67 Byte;

idxGroup = 16#F031: Es handelt es sich um eine Variable aus dem SPS-Prozessabbild der physikalischen Ausgänge;

idxOffset = 16#5: Die Variable liegt auf dem Byteoffset Null und dem Bitoffset 5;

byteSize = 16#1: Der Variablenwert belegt einen Byte im Speicher;

adsDataType = ADST_BIT: Die Ads-Datentyp-Id;

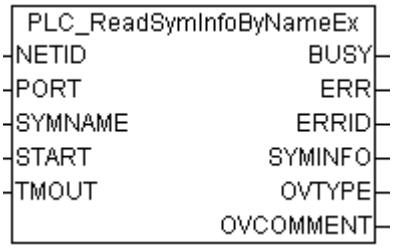
symDataType = BOOL: Die Datentypbezeichnung in der SPS;

symComment = 'STARTS FB EXECUTION': Kommentar, den der Benutzer in der Zeile der Variablendefinition hinzugefügt hat.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.13 PLC_ReadSymInfoByNameEx



Der Funktionsbaustein "PLC_ReadSymInfoByNameEx" hat eine ähnliche Funktionalität wie der Funktionsbaustein PLC_ReadSymInfoByName [▶ 137]. Beide Funktionsbausteine können Symbolinformationen anhand des Symbolnamens auslesen. Der Unterschied bei diesen zwei Bausteinen besteht darin, dass der hier beschriebene Baustein bei einer Überschreitung der verfügbaren Puffergröße keinen Fehler meldet und die nicht ganz kompletten Informationen am Ausgang trotzdem ausgibt. In diesem Fall kann der Kommentar und/oder die Datentypbezeichnung abgeschnitten worden sein. Zwei zusätzliche Ausgangsvariablen zeigen es dies an: *OVTYPE* und *OVCOMMENT*, so dass die Applikation darauf reagieren kann..

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    PORT        :T_AmsPort;
    SYMNAME     :T_MaxString;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT Rechners angegeben werden, auf dem Funktion ausgeführt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

PORT: Die Portnummer des SPS-Laufzeitsystems zu dem die Symbolvariable gehört. Das erste SPS-Laufzeitsystem besitzt die Portnummer 801.

SYMNAME: Der Symbolname der SPS-Variablen deren Informationen gelesen werden soll (max. 255 Zeichen, inklusive des gesamten Pfades z.B. 'MAIN.INIT_TASK.VARINT').

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

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: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

SymInfo: Struktur [▶ 233] mit zusätzlichen Informationen zu der Symbolvariablen.

OVTYPE: Zeigt an ob der String mit der Datentypbezeichnung einen Überlauf verursacht hat (TRUE). Der String mit der Datentypbezeichnung wurde möglicherweise abgeschnitten.

OVCOMMENT: Zeigt an ob der String mit dem Symbolkommentar einen Überlauf verursacht hat(TRUE). Der String mit dem Kommentar wurde möglicherweise abgeschnitten.

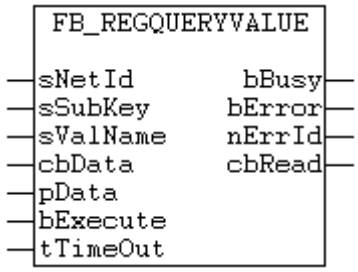
Beispiel für einen Aufruf in FUP

Siehe in der Dokumentation des Funktionsbausteins: PLC_ReadSymInfoByName [▶ 137]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86)	TcUtilities.Lib

4.14 FB_RegQueryValue



Die Systemregistrierung ist ein hierarchisch strukturierter Baum. Ein Knoten im Baum wird als Schlüssel (Key) bezeichnet. Jeder Schlüssel kann wiederum Unterschlüssel (SubKeys) und Datenwerte (Values) enthalten. Mit dem Funktionsbaustein "FB_RegQueryValue" können einzelne Datenwerte (Values) aus der Systemregistrierung aus dem Zweig mit dem vordefinierten Handle **HKEY_LOCAL_MACHINE** ausgelesen werden. Beim Erfolg werden *cbData*-Datenbytes in den Puffer mit der Adresse *pData* hineinkopiert. Mit dem Funktionsbaustein können beliebige Value-Typen (z.B. REG_DWORD, REG_SZ) oder Binärdaten mit einer unbegrenzten Bytelänge (REG_BINARY) ausgelesen werden.

Bemerkung:

Die *sSubKey* und *sValueName*-Strings dürfen keine Leerstrings sein!

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: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Systemregistrierung gelesen werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

sSubKey: String mit dem SubKey-Namen.

sValName: String mit dem Value-Namen.

cbData: Anzahl der zu lesenden Value-Datenbytes.

pData: Adresse eines Datenpuffers/Variablen in den/die Value-Daten hineinkopiert werden sollen. Die Adresse kann dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich den Datenpuffer so zu dimensionieren, dass dieser cbData-Datenbytes aufnehmen kann.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeOut: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

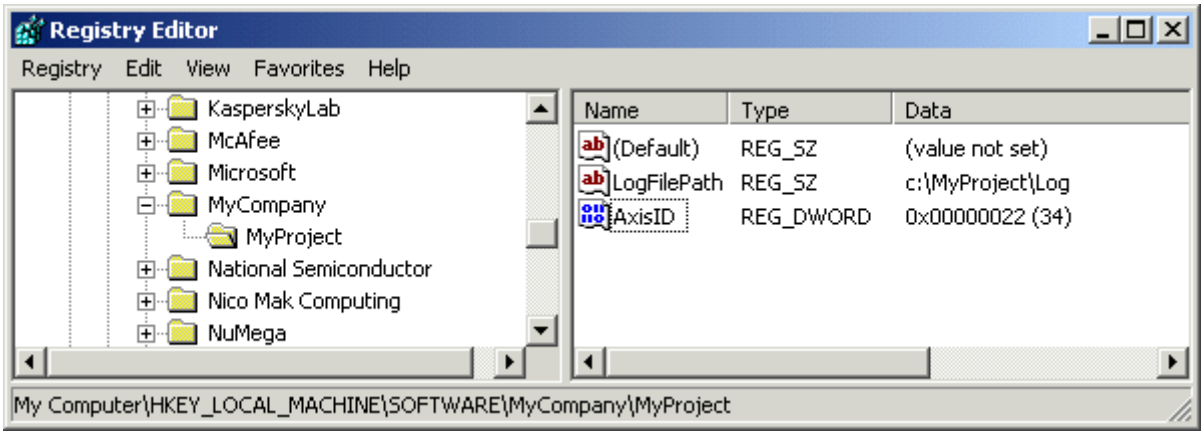
nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer oder einen Befehlsspezifischen Fehlercode zurück (Tabelle).

cbRead: Anzahl der erfolgreich gelesenen Value-Datenbytes.

Fehlercodes	Fehlerbeschreibung
0x00	Kein Fehler
0x01	Der Key mit dem Namen **sSubKey** konnte nicht geöffnet/gefunden werden.
0x02	Der Schlüsselwert mit dem Namen **sValName** konnte nicht geöffnet/gefunden werden.

Beispiele:

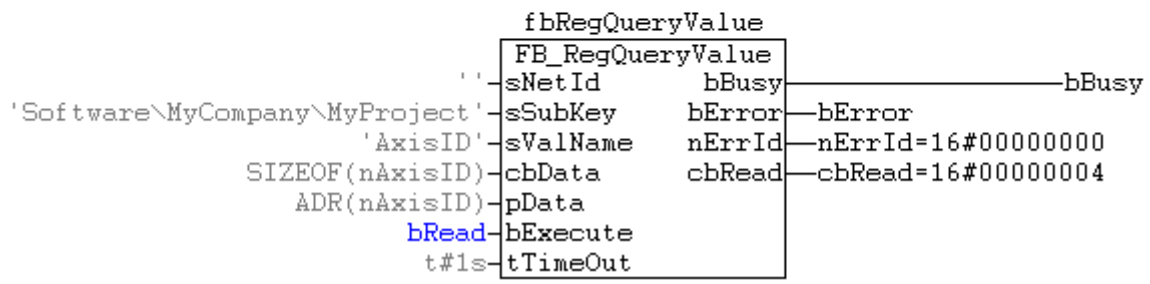
Es sollen aus der Systemregistrierung die Werte *AxisID* und *LogFilePath* gelesen werden.




```
PROGRAM MAIN
VAR
    fbRegQueryValue : FB_RegQueryValue;
    bRead           : BOOL;
    bBusy           : BOOL;
    bError          : BOOL;
    nErrId          : UDINT;
    cbRead          : UDINT;
    sValData        : STRING;
    nAxisID         : DWORD;
END_VAR
```

REG_DWORD-Value lesen:

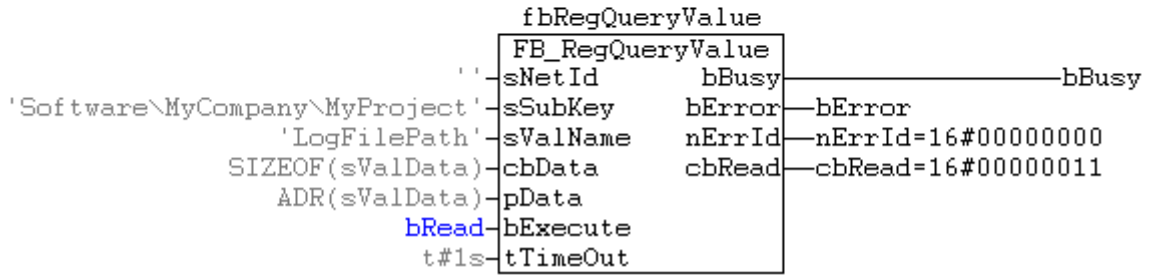
```
fbRegQueryValue
   bRead = TRUE
   bBusy = FALSE
   bError = FALSE
   nErrId = 16#00000000
   cbRead = 16#00000004
   nAxisID = 16#00000022
```



Hier wurde der Wert 0x22 aus der Registrierung in die SPS-Variable *nAxisId* eingelesen.

REG_SZ-Value lesen:

```
fbRegQueryValue
   bRead = TRUE
   bBusy = FALSE
   bError = FALSE
   nErrId = 16#00000000
   cbRead = 16#00000011
   sValData = 'c:\MyProject\Log'
```

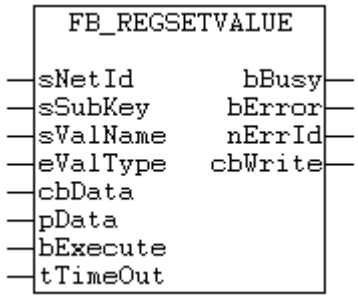


Hier wurde der String 'c:\MyProject\Log' aus der Registrierung in die SPS-Variable *sValData* eingelesen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0 Build > 519	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 Build > 723	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.15 FB_RegSetValue



Die Systemregistrierung ist ein hierarchisch strukturierter Baum. Ein Knoten im Baum wird als Schlüssel (Key) bezeichnet. Jeder Schlüssel kann wiederum Unterschlüssel (SubKeys) und Datenwerte (Values) enthalten.

Mit dem Funktionsbaustein "FB_RegSetValue" können einzelne Schlüsselwerte (Values) oder neue Schlüsselnamen und Werte (SubKeys+Values) in dem Zweig mit dem vordefinierten Handle **HKEY_LOCAL_MACHINE** geschrieben bzw. generiert werden. Es können beliebige Value-Typen (z.B. REG_DWORD, REG_SZ) oder maximal 500 Byte Binärdaten (REG_BINARY) in die Systemregistrierung geschrieben werden. Ist ein Schlüsselwert noch nicht vorhanden, dann wird dieser automatisch neu erzeugt.

Bemerkung:

Die *sSubKey* und *sValueName*-Strings dürfen keine Leerstrings sein!

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: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Systemregistrierung geschrieben werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

sSubKey: String mit dem SubKey-Namen.

sValName: String mit dem Value-Namen.

eValType: Der Datentypformat [▶ 234] der zu schreibenden Registrierungsdaten z.B: REG_DWORD oder REG_SZ.

cbData: Anzahl der zu schreibenden Value-Datenbytes (Bei Stringvariablen inklusiv der abschließenden Null).

pD ata: Adresse von einem Datenpuffer/SPS-Variablen, der/die Value-Daten enthält. Die Adresse kann dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich den Datenpuffer so zu dimensionieren, dass *cbData*-Datenbytes daraus entnommen werden können.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeOut: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer oder den Befehlsspezifischen Fehlercode (Tabelle).

cbWrite: Anzahl der erfolgreich geschriebenen Value-Datenbytes.

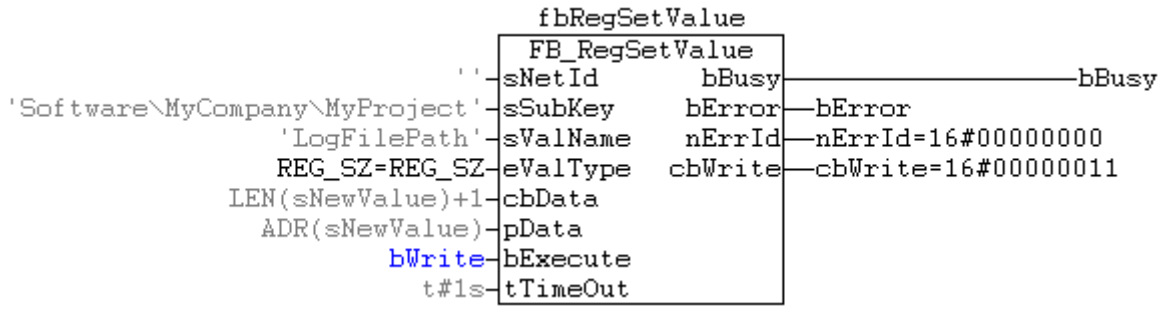
Fehlercodes	Fehlerbeschreibung
0x00	Kein Fehler
0x01	Der Key mit dem Namen **sSubKey** konnte nicht geöffnet/gefunden werden.
0x02	Der Schlüsselwert mit dem Namen **sValName** konnte nicht geöffnet/gefunden werden.

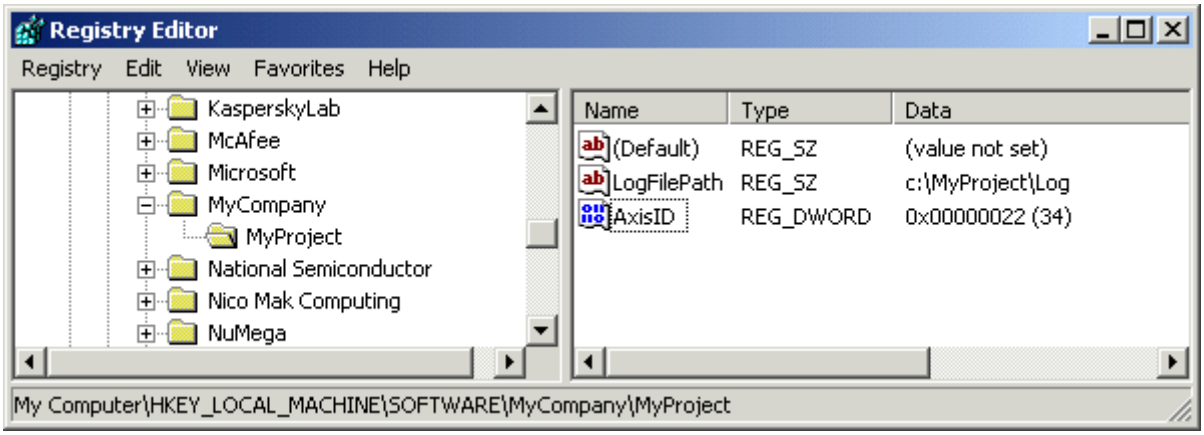
Beispiele:

In dem Zweig mit dem vordefinierten Handle *HKEY_LOCAL_MACHINE* soll ein SubKey *'SOFTWARE\MyCompany\MyProject'* mit dem Schlüsselnamen '*LogFileName'* , dem Typ *REG_SZ*

und Wert '*c:\MyProject\Log"* neu angelegt und gesetzt werden.

```
PROGRAM MAIN
VAR
    fbRegSetValue   : FB_RegSetValue;
    bBusy           : BOOL;
    bError          : BOOL;
    nErrId          : UDINT;
    cbWrite         : UDINT;
    bWrite          : BOOL;
    sNewValue       : STRING := 'c:\MyProject\Log';
END_VAR
```

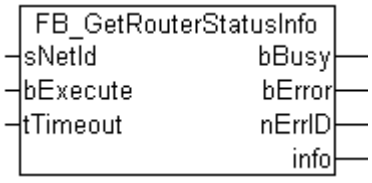




Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0 Build > 519	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 Build > 723	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.16 FB_GetRouterStatusInfo



Mit dem Funktionsbaustein FB_GetRouterStatusInfo können Statusinformationen des TwinCAT Routers aus der SPS ausgelesen werden (verfügbarer Speicher, Anzahl der angemeldeten Ports usw.).

VAR_INPUT

```
VAR_INPUT
    sNetId      : T_AmsNetID := '';
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetId: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen TwinCAT Router-Informationen ausgelesen werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeOut: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

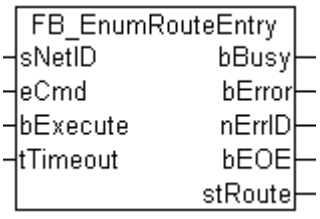
nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

info: Strukturvariable [▶ 244] mit TwinCAT Router-Statusinformationen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1235	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.17 FB_EnumRouteEntry



Mit dem Funktionsbaustein können Informationen über die AMS Router Verbindungen zu anderen TwinCAT Systemen (Remote Routes) ausgelesen werden. Bei mehreren Verbindungen muss der Funktionsbaustein mehrere Male aufgerufen werden und pro Aufruf kann maximal ein Eintrag ausgelesen werden. Mit dem Eingangsparameter *eCmd* kann durch die Liste der Einträge navigiert werden. Der *eCmd*-Eingang bestimmt ob der erste oder nächste Eintrag gelesen werden soll.

VAR_INPUT

```
VAR_INPUT
    sNetID    : T_AmsNetID;
    eCmd      : E_EnumCmdType := eEnumCmd_First;
    bExecute  : BOOL;
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen AMS Router Verbindungen gelesen werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

eCmd: Steuerkommando [▶ 236] für den Aufzählungsbaustein.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    bEOE    : BOOL;
    stRoute : ST_AmsRouteEntry;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

bEOE: Ende der Aufzählung wurde erreicht (end of enumeration). Beim ersten Versuch einen nicht existierenden Eintrag zu lesen wird dieser Ausgang auf TRUE gesetzt. D.h. solange *bEOE* = FALSE und *bError* = FALSE sind, sind auch die gelesenen Einträge gültig.

stRoute: Strukturelement [▶ 244] mit zuletzt gelesenen Verbindungsparametern.

Beispiel in ST:

Auf dem lokalen TwinCAT System sollen die konfigurierten AMS Router Verbindungen ausgelesen und als Meldungen ins TwinCAT System Manager Logger Ausgabe geschrieben werden.

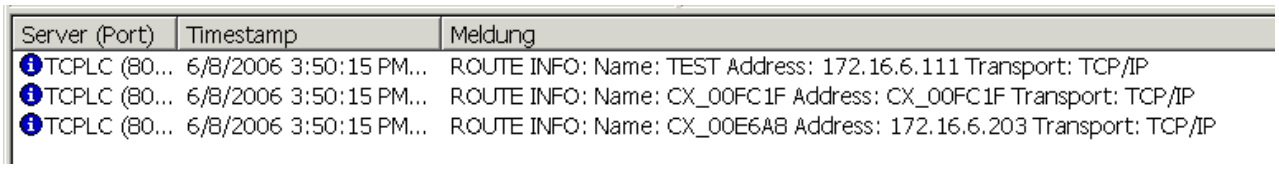
```
PROGRAM P_EnumRouteEntries
VAR
    fbEnum  : FB_EnumRouteEntry := ( sNetID := '', tTimeout := T#5s );
    bEnum   : BOOL := TRUE;
    nState  : BYTE := 0;
    sInfo   : T_MaxString;
END_VAR
```

Bei einer steigenden Flanke an der bEnum-Variablen werden die Verbindungsinformationen ausgelesen.

```
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
```

Die geschriebenen Logmeldungen in der TwinCAT System Manager Logger Ausgabe:

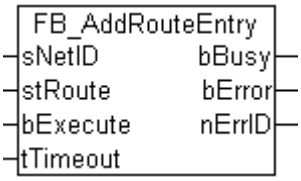


Server (Port)	Timestamp	Meldung
TCPLC (80...	6/8/2006 3:50:15 PM...	ROUTE INFO: Name: TEST Address: 172.16.6.111 Transport: TCP/IP
TCPLC (80...	6/8/2006 3:50:15 PM...	ROUTE INFO: Name: CX_00FC1F Address: CX_00FC1F Transport: TCP/IP
TCPLC (80...	6/8/2006 3:50:15 PM...	ROUTE INFO: Name: CX_00E6A8 Address: 172.16.6.203 Transport: TCP/IP

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.18 FB_AddRouteEntry



Mit dem Funktionsbaustein kann eine neue AMS Router Verbindung (Remote Route) zu einem TwinCAT System hinzugefügt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    stRoute     : ST_AmsRouteEntry;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden auf dem die AMS Router Verbindungsliste um eine neue Verbindung ergänzt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

stRoute: Strukturelement [▶ 244] mit Parametern der neuen Verbindung.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

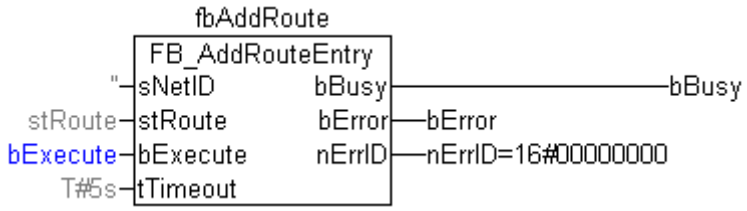
nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

Beispiel in FUP:

Auf dem lokalen TwinCAT System soll eine neue AMS Router Verbindung mit dem Verbindungsnamen: "TEST", TwinCAT Netzwerkadresse: "172.16.6.111.1.1", IP-Adresse: "172.16.6.111" und dem Transportweg: "TCP/IP" hinzugefügt werden.

```
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
```

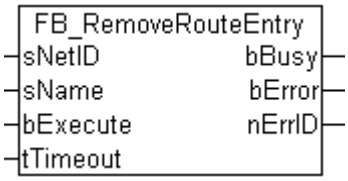
Die benötigten Verbindungsparameter werden bereits im Deklarationsteil initialisiert. Bei einer steigenden Flanke an der bExecute-Variablen wird die neue Verbindung hinzugefügt.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.19 FB_RemoveRouteEntry



Mit dem Funktionsbaustein kann eine vorhandene Verbindung zu einem TwinCAT System aus der Liste der AMS Router Verbindungen (Remote Routes) gelöscht werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    sName       : STRING(MAX_ROUTE_NAME_LEN);
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden auf dem die AMS Router Verbindung gelöscht werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

sName: Verbindungsname der Verbindung die gelöscht werden soll. Die max. Stringlänge ist durch eine Konstante begrenzt (default: 31 Zeichen).

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

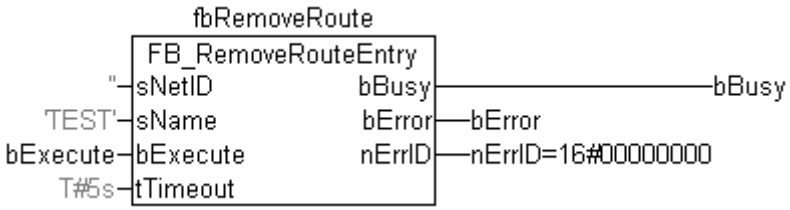
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

Beispiel in FUP:

Die Verbindung mit dem Verbindungsnamen: "TEST" soll aus der Liste der AMS Router Verbindungen auf dem lokalen TwinCAT System gelöscht werden. Bei einer steigenden Flanke an der bExecute-Variablen wird die Verbindung gelöscht.

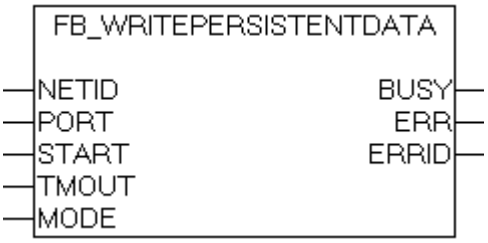
```
PROGRAM P_TEST2
VAR
    fbRemoveRoute   : FB_RemoveRouteEntry;
    bExecute    : BOOL;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.20 FB_WritePersistentData



Bei dem Funktionsbaustein FB_WritePersistentData handelt es sich um eine erweiterte Version des WritePersistentData [▶ 218]-Funktionsbausteins. Über den MODE-Parameter kann aber das Systemverhalten beim Schreiben der pers. Daten [▶ 278] beeinflusst werden (Datenkonsistenz/Taskzykluszeitüberschreitung). Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    PORT        : UINT;
    START       : BOOL;
    TMOUT       : TIME := DEFAULT_ADS_TIMEOUT;
    MODE        : E_PersistentMode;
END_VAR
```

NETID : Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen persistente Daten gespeichert werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

PORT : Der PORT-Parameter bestimmt das Laufzeitsystem dessen persistente Daten gespeichert werden sollen.

START : Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT : Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

MODE : Modus [▶ 235] in dem die pers. Daten geschrieben werden sollen. Siehe: Schreiben der pers. Daten: Systemverhalten [▶ 278].

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

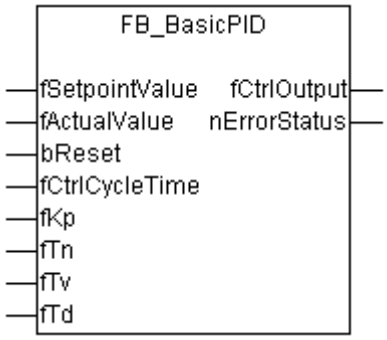
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build >= 959	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.21 FB_BasicPID

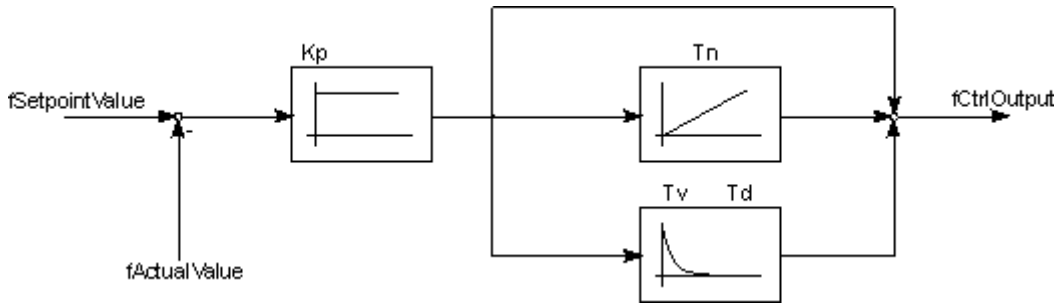


Der Funktionsbaustein stellt ein einfaches diskretisiertes PID-Glied dar.

Übertragungsfunktion:

$$G(s) = K_p(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s})$$

Wirkungsplan:



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 : Sollwert der Regelgröße.

fActualValue : Istwert der Regelgröße.

bReset : Ein TRUE an diesem Eingang setzt die internen Zustandsgrößen sowie den Ausgang des Reglers zurück.

fCtrlCycleTime : Zykluszeit, mit der der Funktionsbaustein aufgerufen wird und der Regelkreis bearbeitet wird [s].

Hier **muss** zwingend die Zykluszeit der SPS-Task angegeben werden, wenn der Baustein in jedem SPS-Zyklus aufgerufen wird,

anderenfalls das entsprechende Vielfache der SPS-Task-Zykluszeit.

fKp : Reglerverstärkung / Reglerbeiwert

fTn : Nachstellzeit [s]

fTv : Vorhaltzeit [s]

fTd : Dämpfungszeit [s]

VAR_OUTPUT

```
VAR_OUTPUT
    fCtrlOutput     : LREAL;
    nErrorStatus    : UINT
END_VAR
```

fCtrlOutput : Ausgang des PID-Gliedes.

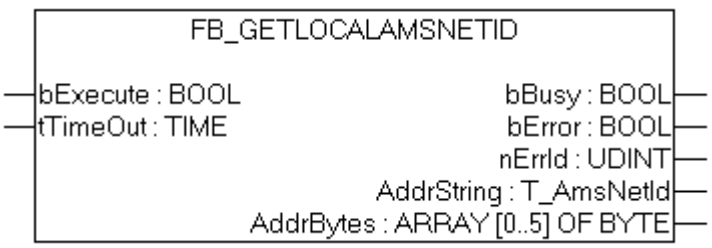
nErrorStatus : Liefert die Fehlernummer, wenn ein Fehler vorliegt (nErrorStatus <> 0).

0 = nERR_NOERROR : Kein Fehler.
1 = nERR_INVALIDPARAM : Ungültige Parameter
2 = nERR_INVALIDCYCLETIME : Ungültige Zykluszeit.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0 Build > 519	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 Build > 739	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.22 FB_GetLocalAmsNetId



Mit dem Funktionsbaustein kann die Netzwerkadresse (AmsNetId) des lokalen TwinCAT PCs ausgelesen werden.

VAR_INPUT

```
VAR_INPUT
    bExecute    :BOOL;
    tTimeOut    :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeOut: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    nErrId      :UDINT;
    AddrString      :T_AmsNetId;
    AddrBytes       :T_AmsNetIdArr;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

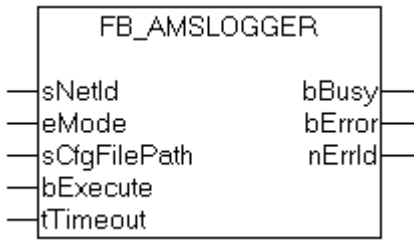
AddrString: Die AmsNetId des lokalen PCs als String.

AddrBytes: Die AmsNetId des lokalen PCs als Byte-Array.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0 Build > 744 TwinCAT v2.9.0 Build > 941	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.23 FB_AmsLogger



Der "TwinCAT AMS Logger" gehört zu den Komponenten des "TwinCAT ADS Monitors" (.. \TwinCAT\AdsMonitor\Logger**TcAmsLog.exe**). Der Logger zeichnet die AMS/ADS-Kommandos auf den Datenträger auf. Die Aufzeichnung kann später z.B. bei einer Fehlersuche mit dem "TwinCAT AMS ADS Viewer" angezeigt und analysiert werden.
Mit dem FB_AmsLogger-Funktionsbaustein kann die Aufzeichnung aus einem SPS-Programm gestartet bzw. gestoppt werden. Der FB_AmsLogger-Funktionsbaustein kann nur mit einer bereits existierenden/ laufenden Instanz der TcAmsLog.exe kommunizieren. D.h. die TcAmsLog.exe muss vorher (z.B. manuell aus dem Start-Menue) oder mit Hilfe des NT_StartProcess [▶ 131]-Bausteins gestartet worden sein.

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: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem der Status des "TwinCAT AMS Loggers" geändert werden soll. Für den Logger auf dem lokalen Rechner kann auch ein Leerstring angegeben werden.

eMode: Der neue Status [▶ 235] in den der "TwinCAT AMS Logger" versetzt werden soll (Aufzeichnung starten/stoppen).

sCfgFilePath: (Optional) Pfad für eine "TwinCAT AMS Logger"-Konfigurationsdatei. Zur Zeit noch nicht implementiert und reserviert für zukünftige Anwendungen (geben Sie bitte ein Leerstring an).

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBuse: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Beispiel:

Beim SPS-Programmstart wird auf einem lokalen System eine Instanz von TcAmsLog.exe gestartet. Beim Setzen der bRecord-Variablen auf TRUE wird die Aufzeichnung AMS/ADS-Kommandos gestartet und beim zurücksetzen auf FALSE gestoppt.

Deklarationsteil:

```
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
```

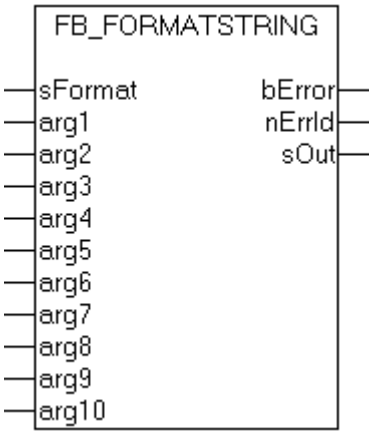
Implementierung:

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build >= 959	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.24 FB_FormatString



Mit dem Funktionsbaustein können bis zu 10 Argumente (ähnlich wie bei fprintf) entsprechend der Formatspezifikation [▶ 275] in einen String konvertiert und formatiert werden. Die Formatierung wird im gleichen SPS-Zyklus durchgeführt. D.h. der Ausgabestring ist sofort nach dem Aufruf des FBs verfügbar.

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: Formatspezifikation als String (z.B. '%+20.5f' oder 'Measure X: %+.10d, Y: %+.10d').

arg1 bis **arg10**: Argumente, die formatiert werden sollen. Folgende Hilfsfunktionen können benutzt werden um SPS-Variablen unterschiedlichsten Typs in den benötigten Datentyp T_Arg [▶ 239] zu konvertieren: F_BYTE [▶ 104], F_WORD [▶ 104], F_DWORD [▶ 105], F_SINT [▶ 105], F_INT [▶ 105], F_DINT [▶ 106], F_USINT [▶ 106], F_UINT [▶ 107], F_UDINT [▶ 107], F_STRING [▶ 108], F_REAL [▶ 103], F_LREAL [▶ 103].

VAR_OUTPUT

```
VAR_OUTPUT
    bError  : BOOL;
    nErrId  : UDINT;
    sOut    : T_MaxString;
END_VAR
```

bError: Sollte ein Fehler bei der Formatierung aufgetreten sein, dann wird dieser Ausgang gesetzt.

nErrId: Liefert bei einem gesetzten bError-Ausgang den Format Fehlercode. [▶ 250]

sOut: Bei Erfolg liefert dieser Ausgang den formatierten Ausgabestring.

Beispiel 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
 );
```

Das Ergebnis:

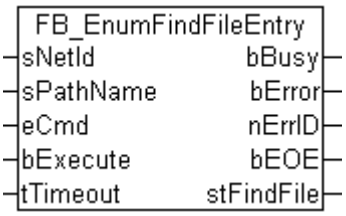
sOut = 'Measure X: +0000000130, Y: +0000000130'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.25 FB_EnumFindFileEntry



Dieser Funktionsbaustein durchsucht ein Verzeichnis nach einer Datei oder nach einem Unterverzeichnis dessen Name dem spezifizierten Namen gleicht. Die gefundenen Einträge können einzeln ausgelesen werden. Siehe auch die Beschreibung des FB_EnumFindFileList [▶ 159]-Funktionsbausteins. Mit dem Eingangsparameter *eCmd* kann durch die Liste der Einträge navigiert werden. Der *eCmd*-Eingang bestimmt z.B. ob der erste oder nächste Eintrag gelesen werden soll.

Hintergrundinformationen:

Eine neue Suche darf nur dann gestartet werden wenn die vorherige Suche komplett abgeschlossen wurde. Für eine komplette Suche muss die Bausteininstanz eventuell mehrere Male aktiviert werden (durch steigende Flanke am bExecute-Eingang). Die Suche ist nur dann komplett abgeschlossen wenn bEOE =TRUE erreicht wurde oder wenn die Suche mit eCmd = eEnumCmd_Abort vorzeitig abgebrochen wurde.

Für das TwinCAT System ist die Suche möglicherweise noch nicht abgeschlossen, wenn von der SPS-Applikation die gesuchte Datei oder das Verzeichnis bereits gefunden wurde.

Wenn nicht alle Einträge ausgelesen werden sollen (d.h. *bEOE=TRUE* wird nicht erreicht), muss der Funktionsbaustein anschließend mit dem Eingangsparameter *eCmd = eEnumCmd_Abort* aufgerufen werden. Dies ist nötig um den Suchvorgang abzuschließen und alle internen Resourcen (Datei-Handles) freizugeben. Wenn *bEOE=TRUE* erreicht wurde, oder ein Fehler auftritt, wird das *eEnumCmd_Abort* intern automatisch ausgeführt.

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: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Verzeichnis durchsucht werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

sPathName: Ein gültiger Verzeichnisname oder Verzeichnis mit Dateinamen als String. Der String kann folgende Platzhalter (* und ?) enthalten. Wenn der Pfad mit einem Platzhalter, Punkt oder dem Verzeichnisnamen endet muss der Benutzer Zugriffsrechte auf diesen Pfad und dessen Unterverzeichnisse haben.

eCmd: Kommandoparameter [▶ 236] für den Aufzählungsbaustein.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

```
    bEOE        : BOOL;
    stFindFile  : ST_FindFileEntry;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

bEOE: Ende der Aufzählung wurde erreicht (end of enumeration). Beim ersten Versuch einen nicht existierenden Eintrag zu lesen wird dieser Ausgang auf TRUE gesetzt. D.h. solange *bEOE* = FALSE und *bError* = FALSE ist, sind auch die gelesenen Einträge gültig.

stFindFile: Bei Erfolg liefert diese Strukturvariable [▶ 244] Informationen zur gefundenen Datei.

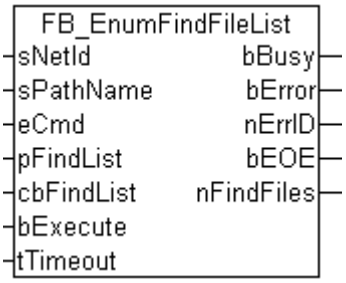
Beispiel:

Siehe: Beispiel: Dateisuche (FB_EnumFindFileEntry, FB_EnumFindFileList). [▶ 255]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1302	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.26 FB_EnumFindFileList



Dieser Funktionsbaustein durchsucht ein Verzeichnis nach einer Datei oder nach einem Unterverzeichnis dessen Name dem spezifizierten Namen gleicht. Die gefundenen Einträge können einzeln ausgelesen werden. Siehe auch die Beschreibung des FB_EnumFindFileEntry [▶ 158]-Funktionsbausteins. Mit dem Eingangsparameter *eCmd* kann durch die Liste der Einträge navigiert werden. Der *eCmd*-Eingang bestimmt z.B. ob der erste oder nächste Eintrag gelesen werden soll.

Hintergrundinformationen:

Eine neue Suche darf nur dann gestartet werden wenn die vorherige Suche komplett abgeschlossen wurde. Für eine komplette Suche muss die Bausteininstanz eventuell mehrere Male aktiviert werden (durch steigende Flanke am bExecute-Eingang). Die Suche ist nur dann komplett abgeschlossen wenn bEOE =TRUE erreicht wurde oder wenn die Suche mit eCmd = eEnumCmd_Abort vorzeitig abgebrochen wurde.

Für das TwinCAT System ist die Suche möglicherweise noch nicht abgeschlossen wenn von der SPS-Applikation die gesuchte Datei oder das Verzeichnis bereits gefunden wurde.

Wenn nicht alle Einträge ausgelesen werden sollen (d.h. *bEOE=TRUE* wird nicht erreicht), muss der Funktionsbaustein anschließend mit dem Eingangsparameter *eCmd* = *eEnumCmd_Abort* aufgerufen werden. Dies ist nötig, um den Suchvorgang abzuschließen und alle internen Resourcen (Datei-Handles) freizugeben. Wenn *bEOE=TRUE* erreicht wurde, oder ein Fehler auftritt, wird das *eEnumCmd_Abort* intern automatisch ausgeführt.

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: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Verzeichnis durchsucht werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

sPathName: Ein gültiger Verzeichnisname oder Verzeichnis mit Dateinamen als String. Der String kann folgende Platzhalter (* und ?) enthalten. Wenn der Pfad mit einem Platzhalter, Punkt oder dem Verzeichnisnamen endet, muss der Benutzer Zugriffsrechte auf diesen Pfad und dessen Unterverzeichnisse haben.

eCmd: Steuerkommando [▶ 236] für den Aufzählungsbaustein.

pFindList: Adresse (Pointervariable) einer Arrayvariablen von Typ: ST_FindFileEntry.

cbFindList: Bytegröße der Arrayvariablen vom Typ: ST_FindFileEntry.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

bEOE: Ende der Aufzählung wurde erreicht (end of enumeration). Beim ersten Versuch einen nichtexistierenden Eintrag zu lesen wird dieser Ausgang auf TRUE gesetzt. D.h. so lange *bEOE* = FALSE und *bError* = FALSE ist, sind auch die gelesenen Einträge gültig.

nFindFiles: Anzahl der gültigen Einträge im Puffer.

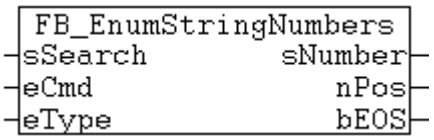
Beispiel:

Siehe: Beispiel: Dateisuche (FB_EnumFindFileEntry, FB_EnumFindFileList). [▶ 255]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1302	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.27 FB_EnumStringNumbers



Mit diesem Funktionsbaustein kann ein String in einer REPEAT- oder WHILE-Schleife nach Zahlen durchsucht werden. Der String kann mehrere Zahlen beinhalten. Die gefundenen Zahlen werden als Teilstrings am Baustein-Ausgang ausgegeben. Es wird ab der aktuellen Position, nach dem ersten Zeichen, das als Zahlenzeichen interpretiert werden kann, gesucht. Die Suche wird beim ersten nicht als Zahl interpretierbaren Zeichen abgebrochen. Der *eCmd*-Parameter bestimmt, ob nach der ersten oder der nächsten Zahl gesucht werden soll. Der *eType*-Parameter bestimmt das Zahlenformat der Zahlen im Suchstring.

VAR_INPUT

```
VAR_INPUT
    sSearch     : T_MaxString;
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    eType       : E_NumGroupTypes := eNumGroup_Float;
END_VAR
```

sSearch: Suchstring in dem nach Zahlen gesucht werden soll.

eCmd: Steuerkommando [▶ 236] für den Aufzählungsbaustein.

eType: Zahlenformat [▶ 237] der gesuchten Zahl. Dieser Parameter bestimmt welche Zeichen ignoriert und welche als Zahlenzeichen interpretiert werden:

Wert	Bedeutung
eNumGroup_Float	Zahlen '0' bis '9', '+', '-' (Vorzeichen) und das 'e' oder 'E' (Exponentzeichen) werden als gültige Zahlenzeichen interpretiert.
eNumGroup_Unsigned	Zahlen '0' bis '9' werden als gültige Zahlenzeichen interpretiert. '+', '-', 'e', 'E'-Zeichen werden als Zahlenzeichen ignoriert.
eNumGroup_Signed	Zahlen '0' bis '9' '+', '-' (Vorzeichen) werden als gültige Zahlenzeichen interpretiert. 'e' und 'E'-Zeichen werden als Zahlenzeichen ignoriert.

Beinhaltet der String z.B. Zahlen in der Exponentialdarstellung dann muss eType = *eNumGroup_Float* gesetzt werden (default).

VAR_OUTPUT

```
VAR_OUTPUT
    sNumber : T_MaxString;
    nPos    : INT;
    bEOS    : BOOL;
END_VAR
```

sNumber: Die zuletzt gefundene Zahl als String.

nPos: Diese Variable liefert immer die Position hinter dem zuletzt gefundenen und richtig formatierten Zahlenzeichen. D.h. an dieser Stelle beginnt der Baustein beim nächsten Aufruf nach neuen Zahlenzeichen zu suchen. *nPos* ist Null wenn die abschließende Null des *sSearch*-Strings erreicht wurde. Das erste Zeichen im String hat die Positionsnummer = 1 (non-zero based position).

bEOS: Diese Variable ist FALSE wenn das Ende des Strings noch nicht erreicht wurde und eine neue Zahl gefunden wurde. In diesem Fall liefert *sNumber* eine gültige Zahl als String. Diese Variable ist TRUE wenn keine weitere Zahl gefunden wurde. Eine weitere Suche muss in diesem Fall abgebrochen werden (*sNumber* liefert keinen gültigen Wert).

Beispiel in ST:

Im folgenden Beispiel wird die *sNumber*-Variable nach gültigen Zahlen durchsucht. Die gefundenen Teilstrings werden in der Array-Variable *arrNums* abgespeichert.

```
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
```

Die gefundenen Strings:

- '-12e-34'
- '-56'
- '+78'

eType-Parameter *eNumGroup_Signed* liefert folgende Ergebnisse:

- '-12'
- '-34'
- '-56'
- '+78'

eType-Parameter *eNumGroup_Unsigned* liefert folgende Ergebnisse:

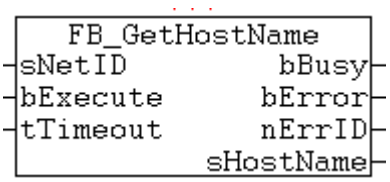
- '12'
- '34'
- '56'
- '78'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.28 FB_GetHostName

Diese Funktionalität ist zur Zeit nur unter dem Betriebssystem Windows CE verfügbar!



Mit diesem Funktionsbaustein kann der Hostname eines TwinCAT PCs ausgelesen werden.

VAR_INPUT

```
VAR_INPUT
    sNetID   : T_AmsNetId;
    bExecute : BOOL;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann die Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Hostname gelesen werden soll. Für den lokalen PC kann auch ein Leerstring angegeben werden.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

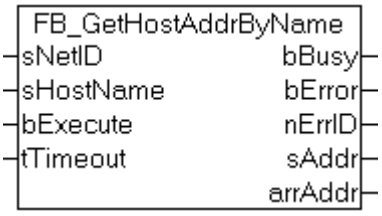
nErrID: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

sHostName: Hostname als String.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.29 FB_GetHostAddrByName



Mit diesem Funktionsbaustein kann die (IPv4) Internet Protokoll Netzwerkadresse für den angegebenen Hostnamen ausgelesen werden. Die Adresse wird als String und Byte-Array zurückgeliefert.

VAR_INPUT

```
VAR_INPUT
    sNetID    : T_AmsNetId;
    sHostName : T_MaxString := '';
    bExecute  : BOOL;
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann die Netzwerkadresse des TwinCAT Rechners angegeben werden, auf dem das Kommando ausgeführt werden soll. Für den lokalen PC (default) kann auch ein Leerstring angegeben werden.

sHostName: Host name als String. Z.B.: 'DataServer1'.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrID  : UDINT;
    sAddr   : T_IPv4Addr := '';
    arrAddr : T_IPv4AddrArr := 0, 0, 0, 0;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrID: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

sAddr: Internet Protokoll Netzwerkadresse (IPv4) als string. Z.B.: '172.16.7.199'

arrAddr: Internet Protokoll Netzwerkadresse als byte array.

Beispiel in ST:

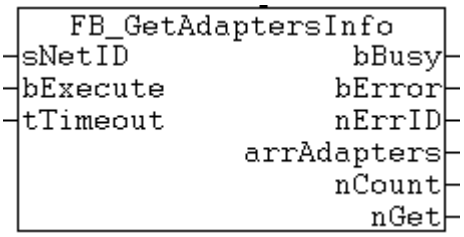
```
PROGRAM MAIN
VAR
    fbGet   : FB_GetHostAddrByName;
    bGet    : BOOL := TRUE;
    bError  : BOOL;
    nErrID  : UDINT;
    sIPv4   : T_IPv4Addr;(* Result: '87.106.8.100' *)
 arrIPv4     : T_IPv4AddrArr;
    state   : BYTE;
END_VAR
```

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1340	PC or CX (x86, ARM)	TcUtilities.Lib

4.30 FB_GetAdaptersInfo



Mit diesem Funktionsbaustein können Adapterinformationen eines TwinCAT PCs ausgelesen werden. Die maximale Anzahl der gelesenen Adapterinformationen ist zur Zeit auf MAX_LOCAL_ADAPTERS + 1 (Default = 6) begrenzt.

VAR_INPUT

```
VAR_INPUT
    sNetID   : T_AmsNetId;
    bExecute : BOOL;
    tTimeout : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen Adapterinformationen gelesen werden sollen. Für den lokalen PC kann auch ein Leerstring angegeben werden.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des Kommandos nicht überschritten werden darf.

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: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrID: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

arrAdapters: Array-Variable mit zuletzt gelesenen Adapterinformationen [▶ 246]. Jedes Arrayelement liefert Informationen eines Adapters.

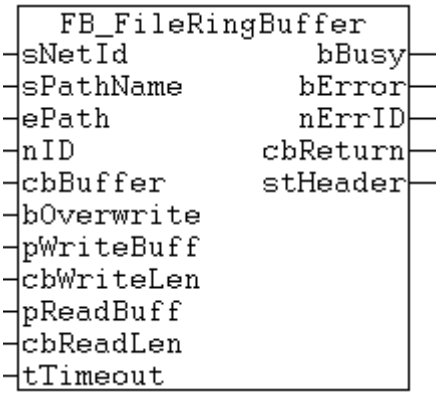
nCount: Maximale Anzahl der gefundenen lokalen Adapter.

nGet: Anzahl der gültigen Einträge der *arrAdapters*-Ausgangsvariablen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.31 FB_FileRingBuffer



Mit dem Funktionsbaustein FB_FileRingBuffer können Datensätze unterschiedlicher Länge in eine Ringpufferdatei geschrieben oder die vorher geschriebenen Datensätze aus der Ringpufferdatei ausgelesen werden. Die geschriebenen Datensätze werden nach dem FIFO-Prinzip in der gleichen Reihenfolge ausgelesen in der sie vorher in die Ringpufferdatei geschrieben wurden. D.h. beim Lesen werden zuerst die ältesten Einträge ausgelesen. Das Öffnen/Schließen/Schreiben/Lesen der Datensätze wird durch Aktionsaufrufe gesteuert. Der Funktionsbaustein besitzt folgende Aktionen:

- **A_Open** (Öffnet eine vorhandene Ringpufferdatei zum Anhängen oder Erzeugen neuer Datensätze. Wenn die Datei bereits geöffnet ist wird kein Fehler zurückgemeldet.)
- **A_Close** (Schließt eine geöffnete Ringpufferdatei. Wenn die Datei bereits geschlossen ist wird kein Fehler zurückgemeldet.)
- **A_Create** (Öffnet eine neue Ringpufferdatei. Wenn die Datei bereits existiert, wird sie überschrieben. Wenn die Datei bereits geöffnet ist wird kein Fehler zurückgeliefert)
- **A_AddTail** (Schreibt einen neuen Datensatz in die Ringpufferdatei.)
- **A_GetHead** (Liest den ältesten Datensatz aus der Ringpufferdatei, entfernt ihn aber nicht (der Dateizeiger wird nicht zum nächsten Datensatz bewegt).)
- **A_RemoveHead** (Liest und entfernt den ältesten Datensatz aus der Ringpufferdatei (der Dateizeiger wird zum nächsten Datensatz bewegt).)
- **A_Reset** (Löscht alle Datensätze in einer geöffneten Ringpufferdatei. Es werden lediglich die Dateizeiger und die Anzahl der Datensätze zurückgesetzt, die vorhandene physikalische Dateigröße wird nicht verändert (die alten Datensätze werden aber mit neuen überschrieben).

Beim Öffnen der Ringpufferdatei wird bei einer existierenden Datei zuerst der Dateiheader gelesen. Bei einer Ringpufferdatei die vorher fehlerfrei geschlossen wurde muss das Bit 0 im Header.status.Bit 0 = Null sein. Bei einem anderen Wert wird angenommen, dass die Datei vorher nicht richtig geschlossen wurde und die korrupte Datei durch eine neue leere Datei ersetzt sowie das Header.status.Bit 1 = 1 gesetzt (file corrupted). Beim Schließen der Datei wird das Header.status.Bit 0 = 0 gesetzt und der komplette Dateiheader in der Datei aktualisiert.

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: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, auf dem die Pufferdatei geschrieben/gelesen werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

sPathName: Enthält den Pfad- und Dateinamen der zu öffnenden Pufferdatei.

Der Pfad kann nur auf das lokale File System des Rechners zeigen! Das bedeutet, Netzwerkpfade können hier nicht angegeben werden!

ePath: Über diesen Eingang kann ein TwinCAT - Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden.

nID: Benutzerdefinierter 32 Bit Wert. Dieser Wert wird beim Öffnen einer neuen Datei in die Datei gespeichert und kann z.B. zur Versionsüberprüfung der Pufferdatei benutzt werden.

cbBuffer: Max. Bytegröße der zu öffnenden Pufferdatei. Dieser Parameter wird beim Erstellen der Datei in dem Dateiheader gespeichert und wird beim erneuten Öffnen derselben Datei überprüft. Sie können nur Dateien erneut öffnen, die mit der gleichen max. Puffergröße erstellt wurden. D.h. Sie können nicht eine Datei mit einer kleineren Puffergröße erstellen, mit Datensätzen füllen und mit einer größeren Puffergröße erneut öffnen. Wenn die Überprüfung der max. Puffergröße fehlschlägt wird automatisch eine neue Datei mit der neuen Puffergröße erstellt und geöffnet. Zusätzlich wird das Bit 1 (file corrupted) in dem Datei-Header-Status gesetzt.

bOverwrite: Schreibverhalten beim Erreichen der max. Dateipuffergröße. Bei TRUE an diesem Eingang werden die ältesten Einträge überschrieben wenn die max. Dateipuffergröße bereits erreicht wurde (es werden so lange Einträge gelöscht bis die freie Puffergröße ausreichend ist um den neuen Eintrag zu speichern). Beim FALSE an diesem Eingang wird ein Pufferüberlauf beim Erreichen der max. Dateipuffergröße als Fehler gemeldet.

pWriteBuff: Adresse der SPS-Variablen oder einer Puffervariablen, die die zu schreibenden Value-Daten enthält. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass *cbWriteLen*-Datenbytes daraus entnommen werden können.

cbWriteLen: Anzahl der zu schreibenden Value-Datenbytes (Bei Stringvariablen inklusive der abschließenden Null).

pReadBuff: Adresse der SPS-Variablen oder einer Puffervariablen in welche die gelesenen Value-Daten hineinkopiert werden sollen. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass diese cbReadLen-Datenbytes aufnehmen kann. Die Bytegröße der Puffervariablen muss größer oder gleich der Größe des zu lesenden Datensatzes sein.

cbReadLen: Anzahl der zu lesenden Value-Datenbytes. Bei einer zu kleinen Puffergröße werden keine Daten kopiert, der Funktionsbaustein meldet einen Puffer-Underflow-Fehler und die benötigte Puffergröße für den nächsten zu lesenden Datensatz wird am *cbReturn*-Ausgang zurückgeliefert.

tTimeOut: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    nErrId      :UDINT;
    cbReturn    :UDINT;
    stHeader    :ST_FileRBufferHead;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer oder den Befehlsspezifischen Fehlercode (Tabelle).

cbReturn: Anzahl der erfolgreich gelesenen Value-Datenbytes. Beim Lesepuffer-Underflow-Fehler liefert dieser Ausgang die benötigte Lesepuffer-Bytegröße.

stHeader: Ringpuffer-Dateiheader [▶ 247]/-Status.

Befehlsspezifischen Fehlercodes	Fehlerbeschreibung
0x8000	Schreiben: Dateipuffer ist leer. Lesen: Dateipuffer-Overflow.
0x8001	SPS-Applikation: Lesepuffer-Underflow (pReadBuff, cbReadLen) ist zu klein dimensioniert.
0x8002	Ringpufferdatei ist geschlossen und muss zuerst geöffnet werden.
0x8003	Falscher Wert des Eingangsparameters.

Die SPS-Applikation muss den binären Aufbau der Datei nicht zwingend kennen. Folgende Grafik zeigt aber den prinzipiellen Aufbau der verwendeten Ringpufferdatei:

Header (48 Byte) siehe in der Beschreibung von ST_FileRBufferHead	Länge Datensatz 1 (4 Byte)	Datensatz 1	Länge Datensatz 2 (4 Byte)	Datensatz 2	Länge Datensatz 3 (4 Byte)	Datensatz 3

Eine leere Ringpufferdatei beinhaltet nur den Header. Hinter dem Header folgt der eigentliche Puffer. Die Header.ptrFirst- und Header.ptrLast-Variable zeigt auf die nächste Position hinter dem Header. Beim Schreiben wird der ptrLast-Dateizeiger nach vorne bewegt. Der ptrFirst-Dateizeiger folgt dem ptrLast-Dateizeiger beim Lesen. Beim Erreichen der max. Puffergröße werden die Zeiger wieder zum Anfang des Puffers bewegt.

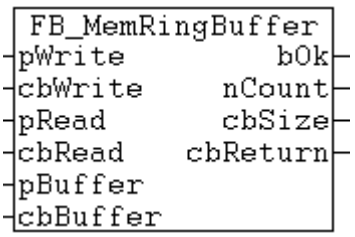
Beispiel:

Siehe: Beispiel: Datei-Ring-Fifo (FB_FileRingBuffer). [▶ 257]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1313	PC oder CX (x86, ARM)	TcUtilities.Lib

4.32 FB_MemRingBuffer



Mit dem Funktionsbaustein FB_MemRingBuffer können Datensätze unterschiedlicher Länge in einen Ringpuffer geschrieben oder die vorher geschriebenen Datensätze aus dem Ringpuffer ausgelesen werden. Die geschriebenen Datensätze werden nach dem FIFO-Prinzip in der gleichen Reihenfolge ausgelesen in der sie vorher in den Ringpuffer geschrieben wurden. D.h. beim Lesen werden zuerst die ältesten Einträge

ausgelesen. Der Pufferspeicher wird dem Funktionsbaustein über die *pBuffer* / *cbBuffer*-Eingangsvariablen zur Verfügung gestellt. Das Schreiben/Lesen der Datensätze wird durch Aktionsaufrufe gesteuert. Der Funktionsbaustein besitzt folgende Aktionen:

- **A_AddTail** (Schreibt einen neuen Datensatz in den Ringpuffer.)
- **A_GetHead** (Liest den ältesten Datensatz aus dem Ringpuffer, entfernt ihn aber nicht.)
- **A_RemoveHead** (Liest und entfernt den ältesten Datensatz aus dem Ringpuffer.)
- **A_Reset** (Löscht alle Datensätze im Ringpuffer.)

VAR_INPUT

```
VAR_INPUT
    pWrite   : DWORD;
    cbWrite  : UDINT;
    pRead    : DWORD;
    cbRead   : UDINT;
    pBuffer  : DWORD;
    cbBuffer : UDINT;
END_VAR
```

pWrite: Adresse der SPS-Variablen oder einer Puffervariablen, die die zu schreibenden Value-Daten enthält. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass *cbWrite*-Datenbytes daraus entnommen werden können.

cbWrite: Anzahl der zu schreibenden Value-Datenbytes (Bei Stringvariablen inklusive der abschließenden Null).

pRead: Adresse der SPS-Variablen oder einer Puffervariablen in welche die gelesenen Value-Daten hineinkopiert werden sollen. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass diese cbRead-Datenbytes aufnehmen kann. Die Bytegröße der Puffervariablen muss größer oder gleich der Größe des zu lesenden Datensatzes sein.

cbRead: Anzahl der zu lesenden Value-Datenbytes. Bei einer zu kleinen Puffergröße werden keine Daten kopiert, der Funktionsbaustein meldet einen Puffer-Underflow-Fehler (bOk = FALSE) und die benötigte Puffergröße für den nächsten zu lesenden Datensatz wird am *cbReturn*-Ausgang zurückgeliefert.

pBuffer: Adresse einer SPS-Variablen (z.B. ARRAY[...] OF BYTES) die vom Funktionsbaustein als Pufferspeicher benutzt werden soll. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbBuffer: Maximale Bytegröße der SPS-Variablen die als Pufferspeicher benutzt werden soll. Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     :BOOL;
    nCount  :UDINT;
    cbSize  :UDINT;
    cbReturn:UDINT;
END_VAR
```

bOk: Liefert TRUE wenn ein neuer Datensatz erfolgreich hinzugefügt oder entfernt werden konnte und FALSE beim Puffer-Überlauf oder wenn keine Einträge im Puffer mehr vorhanden sind.

nCount: Liefert die aktuelle Anzahl der gepufferten Datensätze.

cbSize: Liefert die aktuelle Anzahl der belegten Datenbytes im Puffer. Die Anzahl der belegten Datenbytes ist immer größer als die tatsächliche Anzahl der geschriebenen Value-Daten. Jeder Datensatz wird um zusätzliche Informationen ergänzt um ihn später lokalisieren zu können.

cbReturn: Anzahl der erfolgreich gelesenen Value-Datenbytes. Beim Lesepuffer-Underflow-Fehler liefert dieser Ausgang die benötigte Lesepuffer-Bytegröße. In diesem Fall ist die *cbRead*-Länge zu klein dimensioniert.

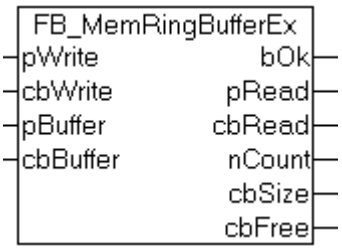
Beispiel:

Siehe: Beispiel: Memory-Ring-Fifo (FB_MemRingBuffer). [▶ 258]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1318	PC oder CX (x86, ARM)	TcUtilities.Lib

4.33 FB_MemRingBufferEx



Mit dem Funktionsbaustein FB_MemRingBufferEx können Datensätze unterschiedlicher Länge in einen Ringpuffer geschrieben oder die vorher geschriebenen Datensätze aus dem Ringpuffer ausgelesen werden. Die geschriebenen Datensätze werden nach dem FIFO-Prinzip in der gleichen Reihenfolge ausgelesen in der sie vorher in den Ringpuffer geschrieben wurden. D.h. beim Lesen werden zuerst die ältesten Einträge ausgelesen. Der Pufferspeicher wird dem Funktionsbaustein über die *pBuffer* / *cbBuffer*-Eingangsvariablen zur Verfügung gestellt. Das Schreiben/Lesen der Datensätze wird durch Aktionsaufrufe gesteuert.

Dieser Funktionsbaustein ähnelt in der Funktion dem FB_MemRingBuffer [▶ 168]-Funktionsbaustein. Beim Lesen der Datensätze kopiert der FB_MemRingBuffer die Daten in eine externe Puffervariable. Der FB_MemRingBufferEx liefert aber nur eine Referenz auf den Datensatz (Addresspointer/Länge). Die Applikation muss dann die Daten für die Weiterverarbeitung selber umkopieren.

Der Funktionsbaustein besitzt folgende Aktionen:

- **A_AddTail** (Schreibt einen neuen Datensatz in den Ringpuffer.)
- **A_GetHead** (Liefert eine Referenz: Adresspointer/Länge auf den ältesten Datensatz aus dem Ringpuffer, entfernt ihn aber nicht.)
- **A_FreeHead** (Liest und entfernt den ältesten Datensatz aus dem Ringpuffer. Der zurück gelieferte Addresspointer/Länge ist Null! Der freie Speichersegment wird für einen neuen Datensatz freigegeben.)
- **A_Reset** (Löscht alle Datensätze im Ringpuffer.)
- **A_GetFreeSize** (Liefert die Bytegröße des größten freien Speichersegments im Puffer)

VAR_INPUT

```
VAR_INPUT
    pWrite       : DWORD;
    cbWrite      : UDINT;
    pBuffer      : DWORD;
    cbBuffer     : UDINT;
END_VAR
```

pWrite: Adresse der SPS-Variablen oder einer Puffervariablen, die die zu schreibenden Value-Daten enthält. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass *cbWrite*-Datenbytes daraus entnommen werden können.

cbWrite: Anzahl der zu schreibenden Value-Datenbytes (bei Stringvariablen inklusive der abschließenden Null). Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

pBuffer: Adresse einer SPS-Variablen (z.B. ARRAY[...] OF BYTES) die vom Funktionsbaustein als Pufferspeicher benutzt werden soll. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbBuffer: Max. Bytegröße der SPS-Variablen die als Pufferspeicher benutzt werden soll. Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     : BOOL;
    pRead   : DWORD;
    cbRead  : UDINT;
    nCount  : UDINT;
    cbSize  : UDINT;
    cbFree  : UDINT;
END_VAR
```

bOk: Liefert TRUE wenn ein neuer Datensatz erfolgreich hinzugefügt oder entfernt werden konnte und FALSE beim Puffer-Überlauf oder wenn keine Einträge im Puffer mehr vorhanden sind.

pRead: Diese Variable liefert nach dem Aufruf der Aktion: *A_GetHead* beim Erfolg (bOk=TRUE) eine Referenz (Addresspionter) auf den ältesten Datensatz im Ringpuffer. Es wird Null zurückgeliefert wenn keine Datensätze mehr im Ringpuffer vorhanden sind.

cbRead: Diese Variable liefert nach dem Aufruf der Aktion: *A_GetHead* beim Erfolg (bOk=TRUE) die Länge vom ältesten Datensatz im Ringpuffer. Es wird Null zurückgeliefert wenn keine Datensätze mehr im Ringpuffer vorhanden sind.

nCount: Liefert die aktuelle Anzahl der gepufferten Datensätze.

cbSize: Liefert die aktuelle Anzahl der belegten Datenbytes im Puffer. Die Anzahl der belegten Datenbytes ist immer größer als die tatsächliche Anzahl der geschriebenen Value-Daten. Jeder Datensatz wird um zusätzliche Informationen ergänzt um ihn später lokalisieren zu können.

cbFree: Lifert nach dem Aufruf der Aktion: *A_GetFreeSize* die Bytegröße des größten freien Speichersegments im Puffer. Die Datensätze müssen auf kontinuierlichen Adressen im Pufferspeicher vorhanden sein da der Funktionsbaustein Referenz auf die Datensätze zurückliefert. Dies führt automatisch zu Segmentierung am Pufferände. Dieser Speicher kann nicht verwendet werden wenn der neue Datensatz größer ist als der freie Segment am Pufferende.

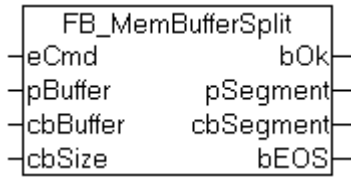
Beispiel:

Siehe: Beispiel: Memory-Ring-Fifo (FB_MemRingBufferEx). [▶ 259]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1332	PC oder CX (x86, ARM)	TcUtilities.Lib

4.34 FB_MemBufferSplit



Dieser Funktionsbaustein teilt einen Speicherbereich (Datenpuffer) in mehrere kleinere Segmente von einer maximal gewünschten Länge auf. Der Funktionsbaustein liefert einen kleineres Teilsegment zurück, falls das letzte Segment eine kleinere Länge als die gewünschte besitzt.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
    cbSize      : UDINT;
END_VAR
```

eCmd: Steuerparameter [▶ 236] für den Aufzählungsbaustein: eEnumCmd_First liefert das erste Segment, eEnumCmd_Next liefert das nächste Segment. Andere Parameter werden nicht benutzt.

pBuffer: Adresse (Pointer) des Datenpuffers, der geteilt werden soll. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbBuffer: Länge des Datenpuffers, der geteilt werden soll. Die Länge kann mit dem SIZEOF-Operator ermittelt werden.

cbSize: Maximale Segmentgröße, in die der Datenpuffer geteilt werden soll.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk         : BOOL;
    pSegment    : DWORD;
    cbSegment   : UDINT;
    bEOS        : BOOL;
END_VAR
```

bOk: TRUE = Success, FALSE = Fehler, falscher Parameterwert oder kein weiteres Segment vorhanden.

pSegment: Adresse (Pointer) auf das nächste Datensegment.

cbSegment: Länge (Bytes) von dem nächsten Datensegment.

bEOS: End of segment. TRUE = Letztes Segment. FALSE = Weitere Segmente folgen.

Beispiel in ST:

Im folgenden Beispiel wird die *buffer*-Variable in 5-Byte-Segmente aufgeteilt. Zu Testzwecken werden die zurück gelieferten Segmente in einen Hexadezimalstring konvertiert.

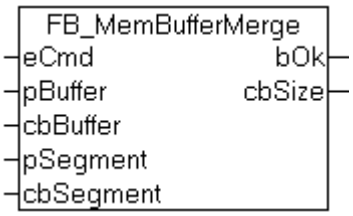
```
PROGRAM MAIN
VAR
    bSplit  : BOOL := TRUE;
    buffer  : ARRAY[1..30] OF BYTE := 16#A,1,2,3,4,5,6,7,8,9,16#B,1,2,3,4,5,6,7,8,9,16#C,1,2,3,4,5,6
,7,8,9;
    fbSplit     : FB_MemBufferSplit;
    sHex    : T_MaxString;
END_VAR
```

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1331	PC or CX (x86, ARM)	TcUtilities.Lib

4.35 FB_MemBufferMerge



Dieser Funktionsbaustein fügt einzelne kleinere Datensegmente zu einem größeren Datensegment zusammen. Der Zielpuffer muss als Eingangsparameter an den Baustein übergeben werden. Es werden keine weiteren Datenbytes hinzugefügt, wenn das Segment, welches hinzugefügt werden soll, die verbliebene freie Puffergröße überschreitet.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
    pSegment    : DWORD := 0;
    cbSegment   : UDINT := 0;
END_VAR
```

eCmd: Steuerparameter [▶ 236] für den Aufzählungsbaustein. eEnumCmd_First fügt das erste Segment, eEnumCmd_Next fügt das nächste Segment hinzu. Andere Parameter werden nicht benutzt.

pBuffer: Adresse (Pointer) auf die Zielpuffervariable. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbBuffer: Maximal verfügbare Größe (in Byte) der Zielpuffervariablen. Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

pSegment: Adresse (Pointer) auf das nächste Datensegment, welches hinzugefügt werden soll (Optional, kann auch Null sein). Die Adresse kann ebenfalls mit dem ADR-Operator ermittelt werden.

cbSegment: Größe des nächsten Datensegments, welches hinzugefügt werden soll (Optional, kann auch Null sein). Die Größe kann ebenfalls mit dem SIZEOF-Operator ermittelt werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk     : BOOL;
    cbSize  : UDINT;
END_VAR
```

bOk: TRUE = Success, FALSE = Pufferüberlauf oder fehlerhafte Eingangsparameter.

cbSize: Der aktuelle Puffer-Füllstatus (Anzahl der Datenbytes im Puffer).

Beispiel in ST:

Im folgenden Beispiel wird zu Testzwecken nach dem Zusammenfügen der kleinen Datensegmente das große Datensegment in einen Hexadezimalstring konvertiert.

```
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;
```

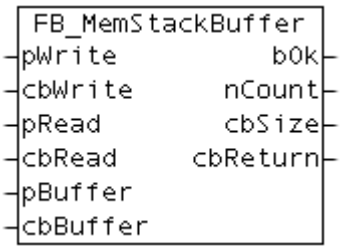
```
    seg2    : ARRAY[0..3] OF BYTE := 6,7,8,9;
    seg3    : ARRAY[0..9] OF BYTE := 10,11,12,13,14,15,16,17,18,19;
    sHex    : T_MaxString;
END_VAR
```

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1331	PC or CX (x86, ARM)	TcUtilities.Lib

4.36 FB_MemStackBuffer



Mit dem Funktionsbaustein FB_MemStackBuffer können Datensätze unterschiedlicher Länge in einen Puffer geschrieben oder die vorher geschriebenen Datensätze aus dem Puffer ausgelesen werden. Die geschriebenen Datensätze werden nach dem LIFO-Prinzip (Last In - First Out) in der umgekehrten Reihenfolge ausgelesen in der sie vorher in den Puffer geschrieben wurden. D.h. beim Lesen werden zuerst die neuesten Einträge ausgelesen. Der Pufferspeicher wird dem Funktionsbaustein über die *pBuffer* und *cbBuffer*-Eingangsvariablen zur Verfügung gestellt. Das Schreiben/Lesen der Datensätze wird durch Aktionsaufrufe gesteuert. Der Funktionsbaustein besitzt folgende Aktionen:

- **A_Push():** Schreibt einen neuen Datensatz in den Puffer;
- **A_Top():** Liest den zuletzt hinzugefügten/neuesten Datensatz aus dem Puffer, entfernt ihn aber nicht;
- **A_Pop():** Liest und entfernt den zuletzt hinzugefügten/neuesten Datensatz aus dem Puffer;
- **A_Reset():** Löscht alle Datensätze im Puffer;

VAR_INPUT

```
VAR_INPUT
    pWrite   : DWORD;
    cbWrite  : UDINT;
    pRead    : DWORD;
    cbRead   : UDINT;
    pBuffer  : DWORD;
    cbBuffer : UDINT;
END_VAR
```

pWrite: Adresse der SPS-Variablen oder einer Puffervariablen, die die zu schreibenden Value-Daten enthält. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass *cbWrite*-Datenbytes daraus entnommen werden können.

cbWrite: Anzahl der zu schreibenden Value-Datenbytes (Bei Stringvariablen inklusive der abschließenden Null).

pRead: Adresse der SPS-Variablen oder einer Puffervariablen in welche die gelesenen Value-Daten hineinkopiert werden sollen. Die Adresse kann mit dem ADR-Operator ermittelt werden. Der Programmierer ist selbst dafür verantwortlich die Puffervariable so zu dimensionieren, dass diese cbRead-Datenbytes aufnehmen kann. Die Bytegröße der Puffervariablen muss größer oder gleich der Größe des zu lesenden Datensatzes sein.

cbRead: Anzahl der zu lesenden Value-Datenbytes. Bei einer zu kleinen Puffergröße werden keine Daten kopiert, der Funktionsbaustein meldet einen Puffer-Underflow-Fehler (bOk = FALSE) und die benötigte Puffergröße für den nächsten zu lesenden Datensatz wird am *cbReturn*-Ausgang zurückgeliefert.

pBuffer: Adresse einer SPS-Variablen (z.B. ARRAY[...] OF BYTES) die vom Funktionsbaustein als Pufferspeicher benutzt werden soll. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbBuffer: Maximale Bytegröße der SPS-Variablen die als Pufferspeicher benutzt werden soll. Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk         :BOOL;
    nCount      :UDINT;
    cbSize      :UDINT;
    cbReturn    :UDINT;
END_VAR
```

bOk: Liefert TRUE wenn ein neuer Datensatz erfolgreich hinzugefügt oder entfernt werden konnte und FALSE beim Puffer-Überlauf oder wenn keine Einträge im Puffer mehr vorhanden sind.

nCount: Liefert die aktuelle Anzahl der gepufferten Datensätze.

cbSize: Liefert die aktuelle Anzahl der belegten Datenbytes im Puffer. Die Anzahl der belegten Datenbytes ist immer größer als die tatsächliche Anzahl der geschriebenen Value-Daten. Jeder Datensatz wird um zusätzliche Informationen ergänzt um ihn später lokalisieren zu können.

cbReturn: Anzahl der erfolgreich gelesenen Value-Datenbytes. Beim Lesepuffer-Underflow-Fehler liefert dieser Ausgang die benötigte Lesepuffer-Bytegröße. In diesem Fall ist die *cbRead*-Länge zu klein dimensioniert.

Beispiel:

Folgendes Beispiel zeigt eine einfache Verwendung des Funktionsbausteins. Es sollen Strings unterschiedlicher Länge gepuffert werden. Die steigende Flanke am *bReset* löscht den Puffer. Wenn Sie *bAdd* = TRUE setzen werden 10 neue Strings in den Puffer geschrieben und beim *bRemove*=TRUE wird der zuletzt geschriebene String aus dem Puffer entfernt. Bei einer steigenden Flanke am *bGet* wird der zuletzt geschriebene String gelesen aber nicht entfernt.

Deklarationsteil:

```
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
```

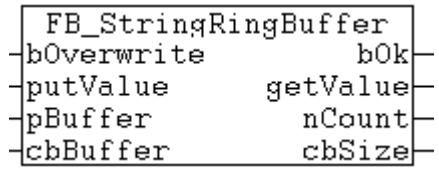
Programmcode:

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 2211	PC oder CX (x86, ARM)	TcUtilities.Lib

4.37 FB_StringRingBuffer



Mit dem Funktionsbaustein FB_StringRingBuffer können String-Variablen in einen Ringpuffer geschrieben oder die vorher geschriebenen String-Variablen aus dem Ringpuffer ausgelesen werden. Die geschriebenen Strings werden nach dem FIFO-Prinzip in der gleichen Reihenfolge ausgelesen in der sie vorher in den Ringpuffer geschrieben wurden. D.h. beim Lesen werden zuerst die ältesten Einträge ausgelesen. Der

Pufferspeicher wird dem Funktionsbaustein über die *pBuffer* / *cbBuffer*-Eingangsvariablen zur Verfügung gestellt. Das Schreiben/Lesen der Strings wird durch Aktionsaufrufe gesteuert. Der Funktionsbaustein besitzt folgende Aktionen:

- **A_AddTail** (Schreibt einen neuen String in den Ringpuffer.)
- **A_GetHead** (Liest den ältesten String aus dem Ringpuffer, entfernt ihn aber nicht.)
- **A_RemoveHead** (Liest und entfernt den ältesten String aus dem Ringpuffer.)
- **A_Reset** (Löscht alle Strings im Ringpuffer.)

Intern wird der FB_MemRingBuffer-Funktionsbaustein verwendet. Sehen Sie auch die Beschreibung des Funktionsbausteins: FB_MemRingBuffer [▶ 168].

VAR_INPUT

```
VAR_INPUT
    bOverwrite  : BOOL;
    putValue    : T_MaxString := '';
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
END_VAR
```

bOverwrite : Beim TRUE und Pufferüberlauf werden die ältesten Einträge überschrieben. Beim FALSE wird beim Pufferüberlauf ein Fehler gemeldet (bOk = FALSE).

putValue : String der in den Ringpuffer geschrieben werden soll.

pBuffer: Adresse einer SPS-Variablen (z.B. ARRAY[...] OF BYTES) die vom Funktionsbaustein als Pufferspeicher benutzt werden soll. Die Adresse kann mit dem ADR-Operator ermittelt werden.

cbBuffer: Max. Bytegröße der SPS-Variablen die als Pufferspeicher benutzt werden soll. Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk      : BOOL;
    getValue : T_MaxString := '';
    nCount   : UDINT;
    cbSize   : UDINT;
END_VAR
```

bOk: Liefert TRUE wenn ein neuer String erfolgreich hinzugefügt oder entfernt werden konnte und FALSE beim Puffer-Überlauf oder wenn keine Einträge im Puffer mehr vorhanden sind.

getValue: Dieser Ausgang liefert den String, der zuletzt aus dem Ringpuffer gelesen wurde.

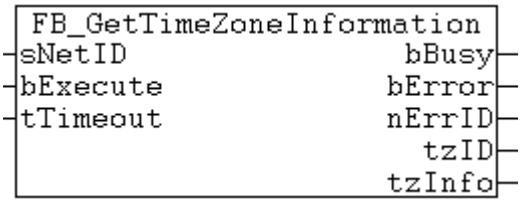
nCount: Liefert die aktuelle Anzahl der gepufferten Strings.

cbSize: Liefert die aktuelle Anzahl der belegten Datenbytes im Puffer. Die Anzahl der belegten Datenbytes ist immer größer als die tatsächliche Anzahl der geschriebenen Value-Daten. Jeder String wird um zusätzliche Informationen ergänzt um ihn später lokalisieren zu können.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC oder CX (x86, ARM)	TcUtilities.Lib

4.38 FB_GetTimeZoneInformation



Mit dem Funktionsbaustein können die Zeitzonen-Einstellungen des Betriebssystems ausgelesen werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Zeitzonen-Einstellungen ausgelesen werden sollen. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
    tzID    : E_TimeZoneID;
    tzInfo  : ST_TimeZoneInformation;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

tzID: Zusätzliche Sommer-/ Winterzeit-Informationen [▶ 238] (nicht immer vorhanden).

tzInfo: Bei Erfolg liefert diese Strukturvariable [▶ 248] die aktuellen Zeitzone-Informationen des Betriebsystems.

Beispiel: Siehe in der Beschreibung des FB_SetTimeZoneInformation [▶ 179]-Funktionsbausteins.

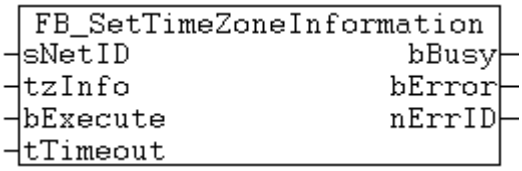
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation, FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1319	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.39 FB_SetTimeZoneInformation



Mit dem Funktionsbaustein können die Zeitzonen-Einstellungen des Betriebssystems geändert bzw. gesetzt werden.

> Das Betriebsystem ändert teilweise die Uhrzeiteinstellungen nachdem die neuen Zeitzonen-Einstellungen gesetzt wurden. Die Uhrzeit muss möglicherweise neu eingestellt werden. Die Uhrzeit kann mit dem Funktionsbaustein: NT_SetLocalTime [▶ 130] eingestellt werden.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetID;
    tzInfo      : ST_TimeZoneInformation;
    bExecute    : BOOL;
    tTimeout    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

sNetID: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Zeitzone-Konfiguration geändert werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

tzInfo: Struktur [▶ 248] mit den neuen Zeitzone-Einstellungen, die gesetzt werden sollen.

bExecute: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

Beispiel in ST:

Auf dem lokalen TwinCAT System soll die Zeitzone: "West Europa Standard Time" eingestellt werden. Als Beispiel wurde in der SPS-Bibliothek bereits eine Konstante: **WEST_EUROPE_TZI** mit den passenden Parameterwerten deklariert. Um andere Zeitzonen konfigurieren zu können muss der *tzInfo*-Eingang des Funktionsbausteins mit entsprechenden Werten belegt werden (siehe in der Beschreibung der ST_TimeZoneInformation [▶ 248]-Struktur).

```
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
```

Der Deklarationsteil:

```
PROGRAM MAIN
VAR
    fbGet : FB_GetTimeZoneInformation;
    fbSet : FB_SetTimeZoneInformation;

    tzi_get : ST_TimeZoneInformation;
    tzID : E_TimeZoneID;

    bGet : BOOL := TRUE;
    bSet : BOOL := FALSE;
END_VAR
```

Bei einer steigenden Flanke an der *bSet* -Variablen wird die gewünschte Zeitzonen-Einstellung gesetzt. Zur Kontrolle können die aktuellen Einstellungen mit einer steigenden Flanke an der *bGet*-Variablen ausgelesen werden.

```
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
```

Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation, NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

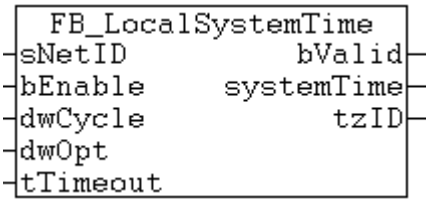
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1319	PC or CX (x86) CX (ARM)	TcUtilities.Lib

Sehen Sie dazu auch

- E_TimeZoneID [▶ 238]

4.40 FB_LocalSystemTime



In einigen Anwendungen wird die lokale Windows-Systemzeit mit Hilfe des SNTP-Zeitservers oder einer Funkuhr synchronisiert. Oft muss die lokale Windows-Systemzeit (z.B. als Zeitstempel Log-Meldungen an das HMI) in der SPS verwendet werden (die lokale Windows-Systemzeit wird in der Taskleiste eingeblendet). Für solche Anwendungen kann der FB_LocalSystemTime-Funktionsbaustein nützlich sein.

Dieser Funktionsbaustein vereint intern die Funktion folgender Funktionsbausteine: RTC_EX2 [▶ 215], NT_GetTime [▶ 129], FB_GetTimeZoneInformation [▶ 177] und NT_SetTimeToRTCTime [▶ 132]. Mit Hilfe des RTC_EX2-Bausteins können z.B. Zeitstempel für Log-Ausgaben generiert werden. Dieser Baustein hat aber den Nachteil, dass seine Uhrzeit nicht synchron mit der lokalen Windows Systemzeit läuft und zyklisch mit dem NT_GetTime-Funktionsbaustein nachsynchronisiert werden muss (siehe RTC-Bausteinbeispiele in der Dokumentation). Die zyklische Synchronisierung der eigenen Uhrzeit (*systemTime*-Ausgang) ist bereits in dem Funktionsbaustein implementiert. Die Zykluszeit ist über den *dwCycle*-Eingang konfigurierbar. Außerdem liefert der Funktionsbaustein Sommerzeit-/ Winterzeit-Zeitzoneninformation.

Der FB_LocalSystemTime-Funktionsbaustein muss zyklisch (z.B. jede Sekunde oder jedem Zyklus der SPS) aufgerufen werden. Dies ist notwendig damit die Uhrzeit zwischen den Synchronisierungen berechnet werden kann.

Bedingt genaue Systemuhr

Die lokale Windows Systemzeit wird mit Hilfe der azyklischen Dienste (ADS-Funktionsbausteine) gelesen. Systembedingt kann die Laufzeit der ADS-Kommandos nicht festgelegt/geschätzt werden. Durch die unterschiedliche Kommandolaufzeiten und abhängig vom Betriebssystem, Synchronisations-Intervall und der Zykluszeit der SPS kann die Uhrzeit am systemTime-Ausgang jittern. Aus diesem Grund ist die vom Baustein gelieferte Uhrzeit nur bedingt für genauere Messaufgaben geeignet. Die Genauigkeit ist aber z.B. ausreichend für Anwendungen aus dem Bereich der Gebäudeautomatisierung.

Sommerzeit/Winterzeit-Umschaltung

Der Funktionsbaustein kann nicht exakt zu dem Zeitpunkt aufgerufen werden bei dem die Umschaltung von Sommer- auf Winterzeit bzw. umgekehrt stattfindet. Um aufwendige Berechnungen zu vermeiden wurde folgende Implementierung gewählt (am Beispiel erklärt).

In unserem Beispiel synchronisiert der Funktionsbaustein z.B. alle 60 Sekunden die eigene Uhrzeit mit der lokalen Windows Systemzeit (grau).
Die SPS-Applikation benötigt und liest die Uhrzeit am Funktionsbaustein z.B. alle 30 Sekunden (blau). In unserem Beispiel wird die Sommer/Winterzeit-Umschaltung mit einer Verspätung von 15 Sekunden erkannt. Dieses Verhalten dürfte aber für die meisten Applikationen unproblematisch sein.

Umschaltung Winterzeit -> Sommerzeit

- ...
- 30-03-2008-01:58:10, tzID = Winterzeit
- 30-03-2008-01:58:15, nach internen Synchronisation
- 30-03-2008-01:58:40, tzID = Winterzeit
- 30-03-2008-01:59:10, tzID = Winterzeit
- 30-03-2008-01:59:15, nach internen Synchronisation
- 30-03-2008-01:59:40, tzID = Winterzeit
- 30-03-2008-02:00:00, das Betriebssystem stellt die Zeit von 2 Uhr auf 3 Uhr um
- 30-03-2008-02:00:10, tzID = Winterzeit (immer noch!)
- 30-03-2008-03:00:15, nach internen Synchronisation, folgende tzID = Sommerzeit
- 30-03-2008-03:00:40, tzID = Sommerzeit
- 30-03-2008-03:01:10, tzID = Sommerzeit
- 30-03-2008-03:01:15, nach internen Synchronisation
- 30-03-2008-03:01:40, tzID = Sommerzeit
- ...

Umschaltung Sommerzeit -> Winterzeit

- ...
- 26-10-2008-02:58:10, tzID = Sommerzeit
- 26-10-2008-02:58:15, nach internen Synchronisation
- 26-10-2008-02:58:40, tzID = Sommerzeit

- 26-10-2008-02:59:10, tzID = Sommerzeit
- 26-10-2008-02:59:15, nach internen Synchronisation
- 26-10-2008-02:59:40, tzID = Sommerzeit
- 26-10-2008-03:00:00, das Betriebssystem stellt die Zeit von 3 Uhr auf 2 Uhr um
- 26-10-2008-03:00:10, tzID = Sommerzeit (immer noch!)
- 26-10-2008-02:00:15, nach internen Synchronisation, folgende tzID = Winterzeit
- 26-10-2008-02:00:40, tzID = Winterzeit
- 26-10-2008-02:01:10, tzID = Winterzeit
- 26-10-2008-02:01:15, nach internen Synchronisation
- 26-10-2008-02:01:40, tzID = Winterzeit
- ...

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: Hier kann ein String mit der Netzwerkadresse des TwinCAT-Rechners angegeben werden, dessen Uhrzeit für die Synchronisation benutzt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

bEnable: Bei einer steigenden Flanke an diesem Eingang wird die sofortige Synchronisation der eigenen Uhrzeit mit der lokalen Windows Systemzeit ausgelöst. Der Ausgang *bValid* wird solange auf FALSE gesetzt bis die Synchronisation abgeschlossen wurde. Durch die erste steigende Flanke wird die zyklische Synchronisation aktiviert. D.h. die danach folgenden zyklischen Synchronisierungen werden automatisch ausgeführt. Die Applikation muss in den meisten Fällen nur ein Mal diesen Eingang auf TRUE setzen.

dwCycle: Zykluszeit (in Sekunden) in der der Funktionsbaustein die eigene Uhrzeit nachsynchronisiert. Die zyklische Synchronisation wird nach der ersten steigenden Flanke am *bEnable*-Eingang aktiviert. Default: Synchronisation alle 5 Sekunden.

dwOpt: Zusätzliche Optionsparameter. Zur Zeit stehen folgende Parameter zur Verfügung:

- Bit 0: Wenn gesetzt dann wird zusätzlich die Windows Systemzeit zyklisch zu der Hardware-Uhr (RTC) synchronisiert (entspricht der Funktion: NT_SetTimeToRTCTime). Default: Aktiviert.
  Auf einem Windows CE-System hat diese Option keine Bedeutung.

tTimeout: Gibt die Timeout-Zeit an, die bei der Ausführung der internen ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    bValid      : BOOL;
    systemTime  : TIMESTRUCT;
    tzID        : E_TimeZoneID := eTimeZoneID_Invalid;
END_VAR
```

bValid: Die Uhrzeit am *systemTime*-Ausgang ist ungültig wenn dieser Ausgang FALSE ist. Beim TRUE ist die Uhrzeit gültig (wurde mindestens ein Mal mit der lokalen Windows Zeit synchronisiert).

systemTime: Lokale Windows-Systemzeit [▶ 231].

tzID: Zeitzonen-Information [▶ 238] (Sommerzeit, Winterzeit).

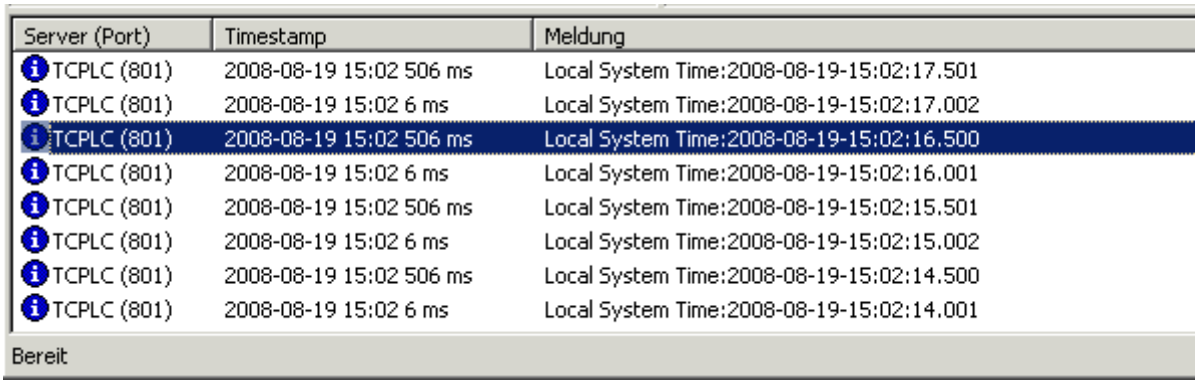
Beispiel in ST:

Im Beispiel wird der FB_LocalSystemTime-Funktionsbaustein beim Programmstart aktiviert (steigende Flanke am *bEnable*-Eingang). Nachdem die Uhrzeit synchronisiert wurde (*bValid* = TRUE) schreibt die SPS alle 500 ms eine Meldung ins TwinCAT System Logview. Die interne Synchronisierung wird jede Sekunde durchgeführt.

```
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
```

Die geschriebenen Meldungen können Sie im TwinCAT System Manager Logview sehen.



Server (Port)	Timestamp	Meldung
TCPLC (801)	2008-08-19 15:02 506 ms	Local System Time:2008-08-19-15:02:17.501
TCPLC (801)	2008-08-19 15:02 6 ms	Local System Time:2008-08-19-15:02:17.002
TCPLC (801)	2008-08-19 15:02 506 ms	Local System Time:2008-08-19-15:02:16.500
TCPLC (801)	2008-08-19 15:02 6 ms	Local System Time:2008-08-19-15:02:16.001
TCPLC (801)	2008-08-19 15:02 506 ms	Local System Time:2008-08-19-15:02:15.501
TCPLC (801)	2008-08-19 15:02 6 ms	Local System Time:2008-08-19-15:02:15.002
TCPLC (801)	2008-08-19 15:02 506 ms	Local System Time:2008-08-19-15:02:14.500
TCPLC (801)	2008-08-19 15:02 6 ms	Local System Time:2008-08-19-15:02:14.001

Bereit

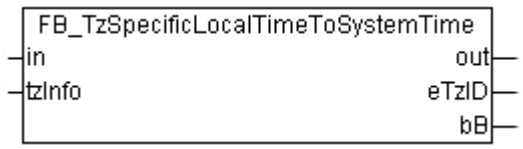
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86) CX (ARM)	TcUtilities.Lib

4.41 FB_TzSpecificLocalTimeToSystemTime



Der Funktionsbaustein konvertiert die Lokalzeit (structured system time format) in UTC-Zeit (structured system time format) unter der Berücksichtigung der angegebenen Zeitzoneninformation. Der Funktionsbaustein: FB_TzSpecificLocalTimeToFileTime [▶ 185] besitzt eine ähnliche Funktionalität, mit dem Unterschied, dass er ein anderes Zeitformat (file time format) umrechnet.

Der Baustein eignet sich nur zur Konvertierung von **kontinuierlichen** Lokalzeit-Zeitstempelinformationen. Die Lokalzeitsprünge durch die Sommer-/Winterzeit Umstellung sind zulässig und werden vom Baustein erkannt. Lokalzeiten, die sich beliebig ändern führen zur fehlerhaften Konvertierung. Der Grund: Die zuletzt konvertierte Zeit wird im Baustein intern gespeichert um die Sommerzeit-/Winterzeit-Information und die B-

Zeiten (siehe unten) beim Zurückstellen der Lokalzeit erkennen zu können. Der Baustein besitzt eine Aktion: A_Reset(). Ein Aufruf dieser Aktion setzt die Ausgänge des Bausteins und die intern gespeicherte (zuletzt konvertierte Zeit) auf Null zurück.

Die Sprünge in der Lokalzeit stellen hier ein Problem dar da sie in eine lineare UTC-Zeit konvertiert werden müssen. Es empfiehlt sich daher für Zeitstempelungsaufgaben die (kontinuierliche) UTC-Zeit zu verwenden und diese erst zur optischen Anzeige der Werte (z. B. in einer Visualisierung) in die jeweilige Lokalzeit zu konvertieren.

Weitere Informationen finden Sie in der Dokumentation des FB_TzSpecificLocalTimeToFileTime [▶ 185] Funktionsbausteins.

VAR_INPUT

```
VAR_INPUT
    in     : TIMESTRUCT;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: Lokalzeit [▶ 231] (structured system time format) die konvertiert werden soll.

tzInfo: Strukturvariable [▶ 248] mit der aktuellen Zeitzone-Information des Betriebssystems.

VAR_OUTPUT

```
VAR_OUTPUT
    out    : TIMESTRUCT;
    eTzID  : E_TimeZoneID := eTimeZoneID_Unknown;
    bB     : BOOL;
END_VAR
```

out: Konvertierte UTC-Zeit (strucutred system time format).

eTzID: Zusätzliche Sommer-/ Winterzeit-Information.

bB: TRUE => B-Zeit (Z.B.:**02:05:00 MEZ B**), FALSE => Übrige Zeit (Z.B.: **02:05:00 MESZ A**). Bei einem Sprung der Lokalzeit zurück wird dieser Ausgang gesetzt und beim Überschreiten der doppelten Lokalzeit wieder zurückgesetzt.

Beispiel:

```
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 );
```

Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime, FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

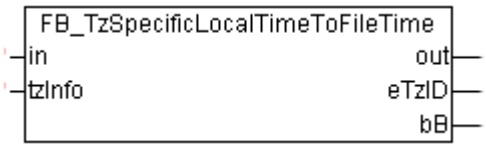
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

- TIMESTRUCT [▶ 231]
- E_TimeZoneID [▶ 238]

4.42 FB_TzSpecificLocalTimeToFileTime

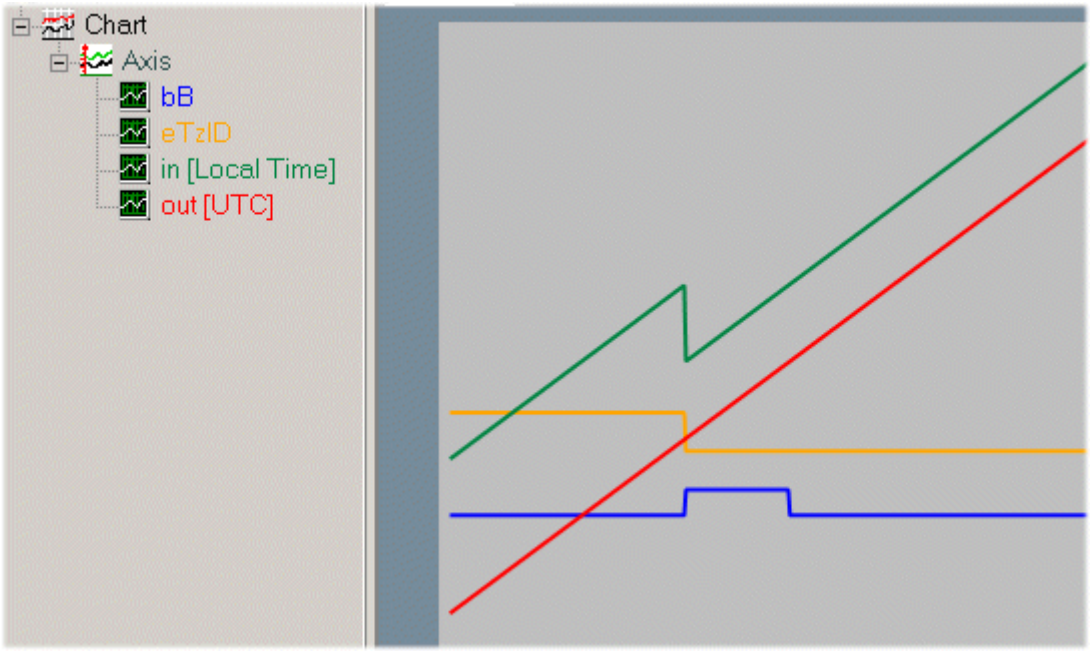


Der Funktionsbaustein konvertiert die Lokalzeit (file time format) in UTC-Zeit (file time format) unter der Berücksichtigung der angegebenen Zeitzoneninformation. Der Funktionsbaustein: FB_TzSpecificLocalTimeToSystemTime [▶ 183] besitzt eine ähnliche Funktionalität, mit dem Unterschied dass er ein anderes Zeitformat (structured system time format) umrechnet.

Der Baustein eignet sich nur zur Konvertierung von **kontinuierlichen** Lokalzeit-Zeitstempelinformation. Die Lokalzeitsprünge durch die Sommer-/Winterzeit Umstellung sind zulässig und werden vom Baustein erkannt. Lokalzeiten, die sich beliebig ändern führen zur fehlerhaften Konvertierung. Der Grund: Die zuletzt konvertierte Zeit wird im Baustein intern gespeichert um die Sommerzeit-/Winterzeit-Information und die B-Zeiten (siehe unten) beim Zurückstellen der Lokalzeit erkennen zu können. Der Baustein besitzt eine Aktion: A_Reset(). Ein Aufruf dieser Aktion setzt die Ausgänge des Bausteins und die intern gespeicherte (zuletzt konvertierte Zeit) auf Null zurück.

Die Sprünge in der Lokalzeit stellen hier ein Problem dar da sie in eine lineare UTC-Zeit konvertiert werden müssen. Es empfiehlt sich daher für Zeitstempelungsaufgaben die (kontinuierliche) UTC-Zeit zu verwenden und diese erst zur optischen Anzeige der Werte (z.B. in einer Visualisierung) in die jeweilige Lokalzeit zu konvertieren.

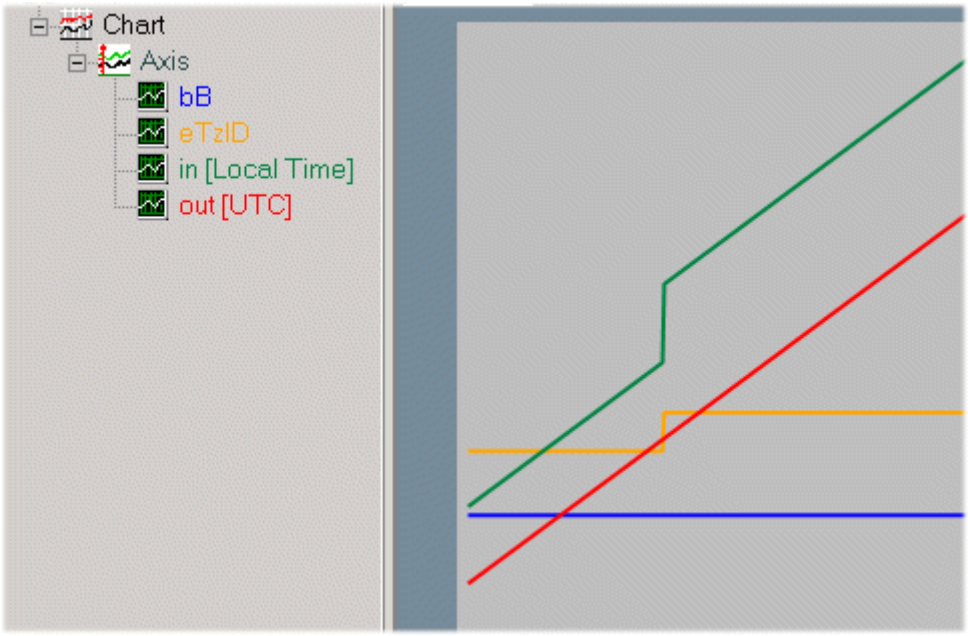
1. Grafische Darstellung des zeitlichen Verhaltens beim Übergang von Sommerzeit zur Winterzeit (hier tzInfo = WEST_EUROPE_TZI):



Die Lokalzeit (grün) macht einen Sprung zurück. Die UTC-Ausgangszeit (rot) verläuft kontinuierlich weiter. Der Lokalzeit: **02h:59m:59s:999ms..** folgt unmittelbar die Zeit: **02h:00m:00s:000ms..** Die Zeiten zwischen 2h und 3h sind zweifach vorhanden. Die doppelte Zeit vor der Zeitumstellung wird z.B. als **02:05:00 MESZ A** bezeichnet und die Zeit nach der Umstellung als **02:05:00 MEZ B**. Die Ausgangsvariable *bB* gibt Information darüber ob es sich um den ersten oder zweiten *Durchgang* handelt. Beim zweiten *Durchgang* ist die *bB*-

Ausgangsvariable (blau) auf TRUE gesetzt. Die *bB*-Ausgangsvariable wird automatisch zurückgesetzt nach dem die doppelte Zeit überschritten wurde. Die Zeitzonen-ID (orange) wechselt von *eTimeZoneID_Daylight* (Sommerzeit) zu *eTimeZoneID_Standard* (Winterzeit).

2. Grafische Darstellung des zeitlichen Verhaltens beim Übergang von Winterzeit zur Sommerzeit (hier tzInfo = WEST_EUROPE_TZI):



Die Lokalzeit (grün) macht einen Sprung nach Vorne. Die UTC-Ausgangszeit (rot) verläuft kontinuierlich weiter. Der Lokalzeit: **2h:59m:59s:999ms..** folgt unmittelbar die Zeit: **3h:00m:00s:000ms..** Die Zeitzonen-ID (orange) wechselt von *eTimeZoneID_Standard* (Winterzeit) zu *eTimeZoneID_Daylight* (Sommerzeit).

VAR_INPUT

```
VAR_INPUT
    in : T_FILETIME;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: Lokalzeit (file time format [▶ 240]) die konvertiert werden soll.

tzInfo: Strukturvariable [▶ 248] mit der aktuellen Zeitzone-Information des Betriebssystems.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : T_FILETIME;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
ND_VAR
```

out: Konvertierte UTC-Zeit (file time format [▶ 240]).

eTzID: Zusätzliche Sommer-/ Winterzeit-Information [▶ 238].

bB: TRUE => B-Zeit (Z.B.:**02:05:00 MEZ B**), FALSE => Übrige Zeit (Z.B.: **02:05:00 MESZ A**). Bei einem Sprung der Lokalzeit zurück wird dieser Ausgang gesetzt und beim Überschreiten der doppelten Lokalzeit wieder zurückgesetzt.

Beispiel:

Die Lokalzeit: DT#2011-09-02-11:01:31 wird in UTC-Zeit: DT#2011-09-02-09:01:31 konvertiert.

```
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 );
```

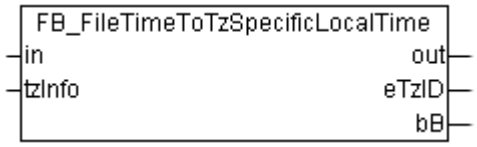
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime, FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

4.43 FB_FileTimeToTzSpecificLocalTime

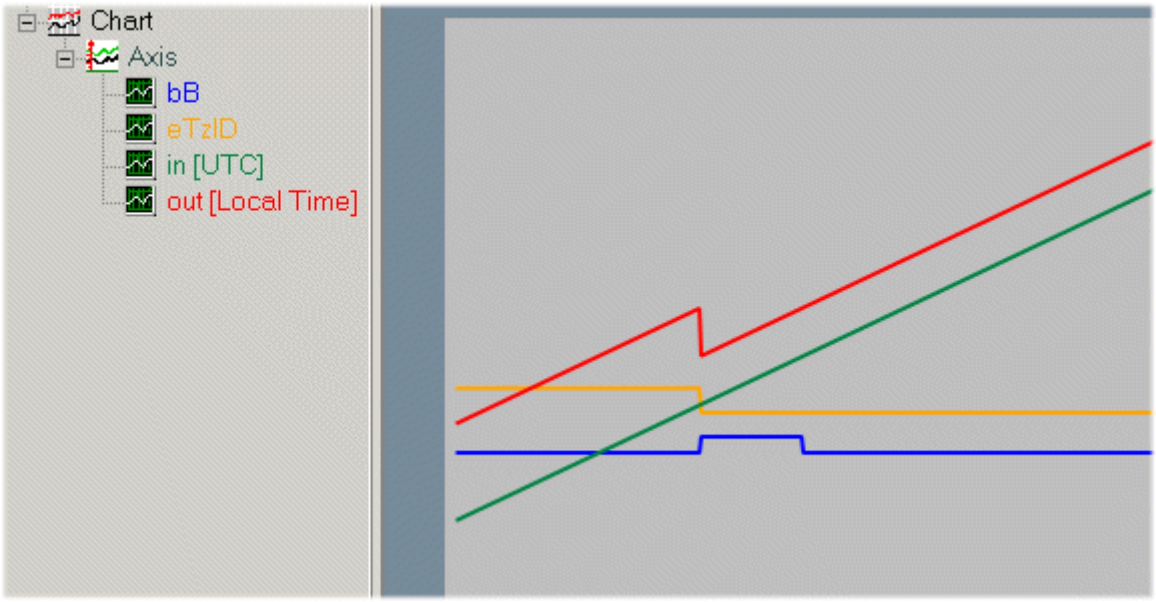


Der Funktionsbaustein konvertiert die UTC-Zeit (file time format) in Lokalzeit (file time format) unter der Berücksichtigung der angegebenen Zeitzoneninformation. Der Funktionsbaustein: FB_SystemTimeToTzSpecificLocalTime [▶ 189] besitzt eine ähnliche Funktionalität, mit dem Unterschied dass er ein anderes Zeitformat (structured system time format) umrechnet.

Der Baustein eignet sich nur zur Konvertierung von **kontinuierlichen** UTC-Zeitstempelinformation. Anhand der Zeitzoneninformation errechnet der Funktionsbaustein die erforderlichen Zeitsprünge in der Lokalzeit (Sommer-/Winterzeit Umstellung). Zeitsprünge in der UTC-Eingangszeit sind nicht zulässig und führen zur fehlerhaften Konvertierung. Der Grund: Die zuletzt konvertierte Zeit wird im Baustein intern gespeichert um aus dem Verlauf der UTC-Eingangszeit und dem gespeicherten Wert die B-Zeiten (siehe unten) beim Zurückstellen der Lokalzeit erkennen zu können.

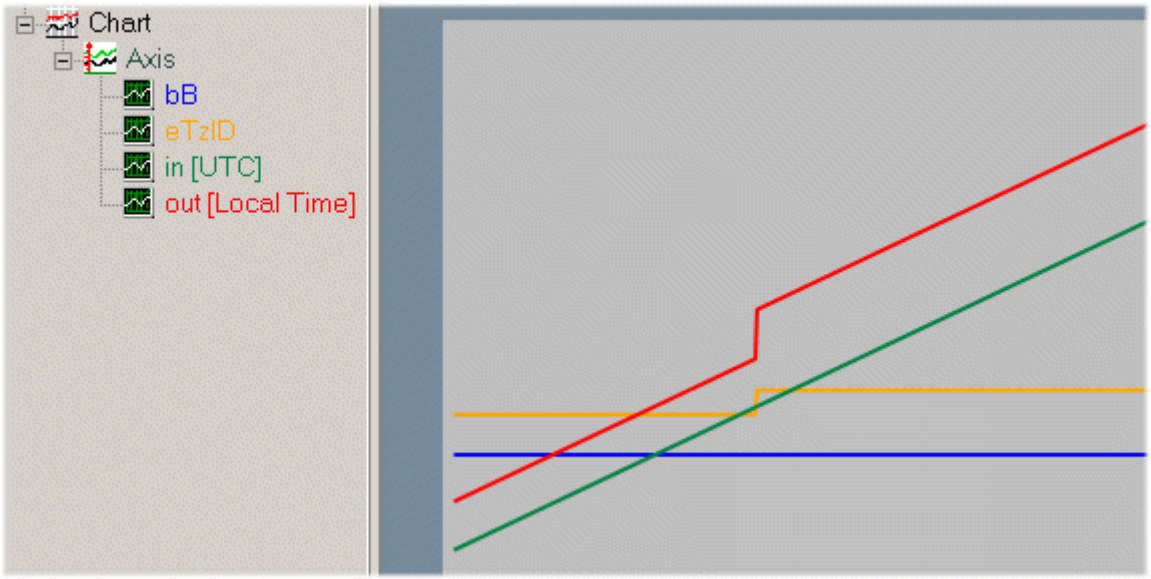
Der Baustein besitzt eine Aktion: A_Reset(). Ein Aufruf dieser Aktion setzt die Ausgänge des Bausteins und die intern gespeicherte (zuletzt konvertierte Zeit) auf Null zurück.

1. Grafische Darstellung des zeitlichen Verhaltens beim Übergang von Sommerzeit zur Winterzeit (hier tzInfo = WEST_EUROPE_TZI):



Die UTC-Eingangszeit (grün) verläuft kontinuierlich. Die Lokalzeit (rot) macht einen Sprung zurück. Der Lokalzeit: **02h:59m:59s:999ms..** folgt unmittelbar die Zeit: **02h:00m:00s:000ms..** Die Zeiten zwischen 2h und 3h sind zweifach vorhanden. Die doppelte Zeit vor der Zeitumstellung wird z.B. als **02:05:00 MESZ A** bezeichnet und die Zeit nach der Umstellung als **02:05:00 MEZ B**. Die Ausgangsvariable *bB* gibt Information darüber ob es sich um den ersten oder zweiten *Durchgang* handelt. Beim zweiten *Durchgang* ist die *bB*-Ausgangsvariable (blau) auf TRUE gesetzt. Die *bB*-Ausgangsvariable wird automatisch zurückgesetzt nachdem die doppelte Zeit überschritten wurde. Die Zeitzonen-ID (orange) wechselt von *eTimeZoneID_Daylight* (Sommerzeit) zu *eTimeZoneID_Standard* (Winterzeit).

2. Grafische Darstellung des zeitlichen Verhaltens beim Übergang von Winterzeit zur Sommerzeit (hier tzInfo = WEST_EUROPE_TZI):



Die UTC-Eingangszeit (grün) verläuft kontinuierlich. Die Lokalzeit (grün) macht einen Sprung nach vorne. Der Lokalzeit: **2h:59m:59s:999ms..** folgt unmittelbar die Zeit: **3h:00m:00s:000ms..** Die Zeitzonen-ID (orange) wechselt von *eTimeZoneID_Standard* (Winterzeit) zu *eTimeZoneID_Daylight* (Sommerzeit).

VAR_INPUT

```
VAR_INPUT
    in     : T_FILETIME;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: UTC-Zeit (file time format [▶ 240]) die konvertiert werden soll.

tzInfo: Strukturvariable [▶ 248] mit der aktuellen Zeitzone-Information des Betriebssystems.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : T_FILETIME;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
END_VAR
```

out: Konvertierte Lokalzeit (file time format [▶ 240]).

eTzID: Zusätzliche Sommer-/ Winterzeit-Information [▶ 238].

bB: TRUE => B-Zeit (Z.B.:**02:05:00 MEZ B**), FALSE => Übrige Zeit (Z.B.: **02:05:00 MESZ A**). Bei einem Sprung der Lokalzeit zurück wird dieser Ausgang gesetzt und beim Überschreiten der doppelten Lokalzeit wieder zurückgesetzt.

Beispiel:

Die UTC-Zeit: DT#2011-09-02-09:01:31 wird in Lokalzeit konvertiert. Das Ergebnis ist: 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 );
```

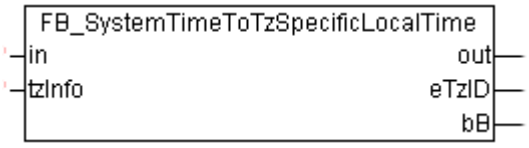
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime [▶ 189], FB_FileTimeTimeToTzSpecificLocalTime, FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

4.44 FB_SystemTimeToTzSpecificLocalTime



Der Funktionsbaustein konvertiert die UTC-Zeit (structured system time format) in Lokalzeit (structured system time format) unter der Berücksichtigung der angegebenen Zeitzoneninformation. Der Funktionsbaustein: FB_FileTimeToTzSpecificLocalTime [▶ 187] besitzt eine ähnliche Funktionalität, mit dem Unterschied dass er ein anderes Zeitformat (file time format) umrechnet.

Der Baustein eignet sich nur zur Konvertierung von **kontinuierlichen** UTC-Zeitstempelinformation. Anhand der Zeitzoneninformation errechnet der Funktionsbaustein die erforderlichen Zeitsprünge in der Lokalzeit (Sommer-/Winterzeit Umstellung). Zeitsprünge in der UTC-Eingangszeit sind nicht zulässig führen zur fehlerhaften Konvertierung. Der Grund: Die zuletzt konvertierte Zeit wird im Baustein intern gespeichert um aus dem Verlauf der UTC-Eingangszeit und dem gespeicherten Wert die B-Zeiten (siehe unten) beim Zurückstellen der Lokalzeit erkennen zu können.

Der Baustein besitzt eine Aktion: A_Reset(). Ein Aufruf dieser Aktion setzt die Ausgänge des Bausteins und die intern gespeicherte (zuletzt konvertierte Zeit) auf Null zurück.

Weitere Informationen finden Sie in der Dokumentation des FB_FileTimeToTzSpecificLocalTime [▶ 187] Funktionsbausteins.

VAR_INPUT

```
VAR_INPUT
    in     : TIMESTRUCT;
    tzInfo : ST_TimeZoneInformation;
END_VAR
```

in: UTC-Zeit (structured system time format [▶ 231]) die konvertiert werden soll.

tzInfo: Strukturvariable [▶ 248] mit der aktuellen Zeitzone-Information des Betriebssystems.

VAR_OUTPUT

```
VAR_OUTPUT
    out   : TIMESTRUCT;
    eTzID : E_TimeZoneID := eTimeZoneID_Unknown;
    bB    : BOOL;
END_VAR
```

out: Konvertierte Lokalzeit (strucutred system time format).

eTzID: Zusätzliche Sommer-/ Winterzeit-Information.

bB: TRUE => B-Zeit (Z.B.:**02:05:00 MEZ B**), FALSE => Übrige Zeit (Z.B.: **02:05:00 MESZ A**). Bei einem Sprung der Lokalzeit zurück wird dieser Ausgang gesetzt und beim Überschreiten der doppelten Lokalzeit wieder zurückgesetzt.

Beispiel:

```
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 );
```

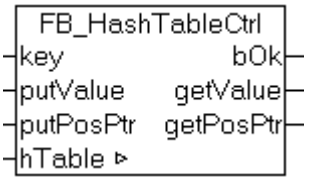
Weitere Zeit-, Zeitzone-Funktionen und -Funktionsbausteine:

FB_TzSpecificLocalTimeToSystemTime [▶ 183], FB_TzSpecificLocalTimeToFileTime [▶ 185], FB_SystemTimeToTzSpecificLocalTime, FB_FileTimeToTzSpecificLocalTime [▶ 187], FB_GetTimeZoneInformation [▶ 177], FB_SetTimeZoneInformation [▶ 179], NT_SetLocalTime [▶ 130], NT_GetTime [▶ 129], NT_SetTimeToRTCTime [▶ 132], F_TranslateFileTimeBias [▶ 61], FB_LocalSystemTime [▶ 180]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 2036	PC or CX (x86, ARM)	TcUtilities.Lib

4.45 FB_HashTableCtrl



Hashtable kann dazu verwendet werden um ein einzelnes Datenelement in einer größeren Menge von Datenelementen schnell auffinden zu können. Die Datenobjekte müssen mit einem eindeutigen Schlüssel versehen werden. Über diesen Schlüssel können die Datenobjekte dann in der Tabelle eindeutig identifiziert und schnell gefunden werden.

Mit dem Funktionsbaustein FB_HashTableCtrl kann eine einfache Hash-Tabelle im SPS-Projekt realisiert werden. Es wird dabei das Verfahren: Hashing mit Verkettung (Separate-Chaining) angewandt.

Die maximale Anzahl der Datenelemente kann zur Laufzeit nicht verändert werden und muss vorher festgelegt werden. Das Hinzufügen/Entfernen/Suchen der Datenelemente wird durch Aktionsaufrufe gesteuert. Der Funktionsbaustein besitzt folgende Aktionen:

- **A_Add** (Fügt ein neues Datenelement der Tabelle hinzu (Schlüssel/Wert). Ein bereits existierendes Element mit dem selben Schlüssel wird überschrieben!)
- **A_GetFirst** (Liest das erste Tabellen-Datenelement. Bei Erfolg liefert *getValue* den dazugehörigen Wert.)
- **A_GetNext** (Liest das nächste Tabellen-Datenelement. Die Adresse: *putPosPtr* muss auf das vorherige Datenelement zeigen!)
- **A_Lookup** (Sucht ein zum Schlüssel (key) passendes Datenelement. Bei Erfolg liefert *getValue* den dazugehörigen Wert.)
- **A_Remove** (Entfernt ein zum Schlüssel passendes Datenelement.)
- **A_RemoveAll** (Entfernt alle Datenelemente)
- **A_RemoveFirst** (Entfernt das erste Datenelement)
- **A_Reset** (Löscht alle Datenelemente und setzt die Tabelle zurück.)
- **A_GetIndexAtPosPtr (ab TwinCAT v2.10 Build >=1339 und TwinCATv2.11 Build >= 1524)** (Liefert den Array-Index des Datenelements an der Adresse: *putPosPtr*. Bei Erfolg liefert *getValue* den Null-Basierten Array-Index. Der Wert *putValue* wird nicht benutzt. Bitte beachten Sie, dass der Wert *getValue* einen Datenelement-Index und nicht den Datenelement-Wert zurückliefert!)

VAR_IN_OUT

```
VAR_IN_OUT
    hTable      : T_HHASHTABLE;
END_VAR
```

hTable: Hash-Tabellen-Handle [▶ 242]. Das Handle muss einmalig vor der Benutzung mit der Funktion: F_CreateHashTableHnd [▶ 66] initialisiert werden. Für jede Tabelle muss eine dazugehörende Instanz der Handle-Variablen angelegt und initialisiert werden.

VAR_INPUT

```
VAR_INPUT
    key             : DWORD := 0;
    putValue        : DWORD := 0;
    putPosPtr       : POINTER TO T_HashTableEntry := 0;
END_VAR
```

key: Schlüssel (32 bit). Über diesen Schlüssel kann ein Datenelement in der Tabelle schnell identifiziert/ gefunden werden.

putValue: Wert/Datenelement (32 bit).

putPosPtr: Adresse auf das Datenelement [▶ 243].

VAR_OUTPUT

```
VAR_OUTPUT
    bOk             : BOOL := FALSE;
    getValue        : DWORD := 0;
    getPosPtr       : POINTER TO T_HashTableEntry := 0;
END_VAR
```

bOk: Liefert TRUE wenn ein neues Datenelement der Tabelle hinzugefügt/entfernt oder in der Tabelle gefunden werden konnte. FALSE wird geliefert, wenn das gesuchte Datenelement nicht gefunden werden konnte, die Tabelle leer ist oder einen Überlauf hat (keine freien Datenelemente mehr).

getValue: Der zum Schlüssel passende Wert.

getPosPtr: Die Adresse auf das Datenelement [▶ 243].

Beispiel:

Siehe: Beispiel: Hash-Tabelle (FB_HashTableCtrl). [▶ 260]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1332	PC oder CX (x86, ARM)	TcUtilities.Lib

4.46 FB_LinkedListCtrl

Mit dem Funktionsbaustein FB_LinkedListCtrl kann eine verkettete Liste im SPS-Projekt realisiert werden. Es wird dabei eine doppelt verkettete Liste erstellt. In einer verketteten Liste können Werte (genannt Knoten) abgespeichert werden. Durch die Liste kann von hinten nach vorne oder umgekehrt iteriert werden. Die Knoten können schnell hinzugefügt oder gelöscht werden.

Die maximale Anzahl der Knoten kann zur Laufzeit nicht verändert werden und muss vor dem Compile-Vorgang festgelegt werden. Als "Knoten-Pool" wird ein Array von Typ: *T_LinkedListEntry* verwendet. Das Hinzufügen/Entfernen/Suchen der Knoten wird durch Aktionsaufrufe gesteuert. Der Funktionsbaustein besitzt folgende Aktionen:

- **A_AddHeadValue** (Fügt einen neuen Knoten mit dem Wert: *putValue* am Anfang der Liste hinzu. Derselbe Wert kann auch mehrfach hinzugefügt werden. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des neuen Knoten.)
- **A_AddTailValue** (Fügt einen neuen Knoten mit dem Wert: *putValue* am Ende der Liste hinzu. Derselbe Wert kann auch mehrfach hinzugefügt werden. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des neuen Knoten.)

- **A_FindNext** (Sucht nach dem nächsten Knoten (relativ zu *putPosPtr*) mit demselben Wert *putValue*. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des Knoten.)
- **A_FindPrev** (Sucht nach dem vorherigen Knoten (relativ zu *putPosPtr*) mit demselben Wert *putValue*. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des Knoten.)
- **A_GetNext** (Navigiert zum nächsten Knoten (relativ zu *putPosPtr*). Die Adresse: *putPosPtr* muss auf den vorherigen Knoten zeigen! Der Wert *putValue* wird nicht benutzt.)
- **A_GetPrev** (Navigiert zum vorherigen Knoten (relativ zu *putPosPtr*) in die entgegengesetzte Richtung wie *A_GetNext*. Die Adresse: *putPosPtr* muss auf den vorherigen Knoten zeigen! Der Wert *putValue* wird nicht benutzt.)
- **A_GetHead** (Liest den Anfangsknoten. Bei Erfolg liefert *getPosPtr* die Adresse des Knoten und *getValue* den dazugehörigen Wert. Der Wert *putValue* und *putPosPtr* wird nicht benutzt.)
- **A_GetTail** (Liest den Endknoten. Bei Erfolg liefert *getPosPtr* die Adresse des Knoten und *getValue* den dazugehörigen Wert. Der Wert *putValue* und *putPosPtr* wird nicht benutzt.)
- **A_RemoveHeadValue** (Entfernt einen Knoten am Anfang der Liste. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des Knoten. Der Wert *putValue* und *putPosPtr* wird nicht benutzt.)
- **A_RemoveTailValue** (Entfernt einen Knoten am Ende der Liste. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des Knoten. Der Wert *putValue* und *putPosPtr* wird nicht benutzt.)
- **A_RemoveValueAtPosPtr** (Sucht und entfernt einen Knoten mit der Adresse: *putPosPtr*. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des Knoten. Der Wert *putValue* wird nicht benutzt.)
- **A_GetIndexAtPosPtr** (Liefert den Array-Index (vom "Knoten-Pool") des Knoten an der Adresse: *putPosPtr*. Bei Erfolg liefert *getValue* den nullbasierten Array-Index. Der Wert *putValue* wird nicht benutzt. Bitte beachten Sie, dass der Wert getValue einen Knoten-Index und nicht den Knoten-Wert zurückliefert!)
- **A_SetValueAtPosPtr** (Aktualisiert/setzt den Wert des Knoten *putValue* an der Adresse *putPosPtr*. Bei Erfolg liefert *getPosPtr* die Adresse und *getValue* den Wert des Knoten.)
- **A_Reset** (Löscht alle Listenelemente und setzt die Liste zurück.)

VAR_IN_OUT

```
VAR_IN_OUT
    hList       : T_HLINKEDLIST;
END_VAR
```

hList: Linked-List-Handle [▶ 243]. Das Handle muss einmalig vor der Benutzung mit der Funktion: F_CreateLinkedListHnd [▶ 67] initialisiert werden. Für jede verkettete Liste muss eine dazugehörende Instanz der Handle-Variablen angelegt und initialisiert werden.

VAR_INPUT

```
VAR_INPUT
    putValue        : DWORD := 0;
    putPosPtr       : POINTER TO T_LinkedListEntry := 0;
END_VAR
```

putValue: Wert/Datenelement (Ausgangsparameter, 32 bit, auch Pointer sind möglich).

putPosPtr: Die Adresse des Knotenelements [▶ 243] (Eingangsparameter).

VAR_OUTPUT

```
VAR_OUTPUT
    bOk         : BOOL := FALSE;
    getValue        : DWORD := 0;
    getPosPtr       : POINTER TO T_LinkedListEntry := 0;
END_VAR
```

bOk: Ergebnis des letzten Aktionsaufrufs. Liefert TRUE wenn ein neues Knotenelement hinzugefügt/entfernt oder in der Liste gefunden werden konnte. FALSE wird geliefert wenn das gesuchte Knotenelement nicht gefunden werden konnte, die Liste leer ist oder einen Überlauf (keine freien Knotenelemente mehr) hat.

getValue: Wert/Datenelement (Ausgangsparameter, 32 bit, auch Pointer sind möglich).

getPosPtr: Die Adresse des Knotenelements [▶ 243] (Ausgangsparameter).

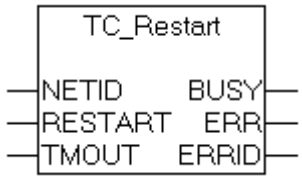
Beispiel:

Siehe: Beispiel: Verkettete-Liste (FB_LinkedListCtrl). [▶ 264]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC oder CX (x86, ARM)	TcUtilities.Lib

4.47 TC_Restart



Mit dem Funktionsbaustein "TC_Restart" kann ein Restart des TwinCAT-Systems durchgeführt werden. Die Funktion entspricht dem Restart-Befehl aus dem TwinCAT-System Menü (rechts in der Windows-Startleiste). Bei einem Restart des TwinCAT-Systems wird zuerst das TwinCAT-System gestoppt und sofort erneut gestartet. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    RESTART     :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem ein Restart des TwinCAT-Systems durchgeführt werden soll. Für ein Restart auf dem lokalen Rechner, kann auch ein Leerstring angegeben werden.

RESTART: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

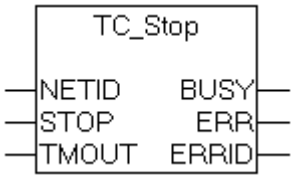
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.48 TC_Stop



Mit dem Funktionsbaustein "TC_Stop" kann ein Stopp des TwinCAT-Systems durchgeführt werden. Die Funktion entspricht dem Stopp-Befehl aus dem TwinCAT-System Menü (rechts in der Windows-Startleiste).

Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    STOP        :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem ein TwinCAT-Systemstopp durchgeführt werden soll. Für ein TwinCAT-Systemstopp auf dem lokalen Rechner, kann auch ein Leerstring angegeben werden.

STOP: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

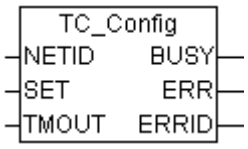
ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.49 TC_Config



Ein TwinCAT System, welches sich im RUN-Modus befindet (grünes TwinCAT System Icon) kann mit dem Funktionsbaustein "TC_Config" in den CONFIG-Modus (blaues TwinCAT System Icon) versetzt werden. Wenn sich das System bereits in dem CONFIG-Modus befindet, dann wird das System zuerst in den STOP-Modus (rotes TwinCAT System Icon) und dann in den CONFIG-Modus versetzt. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    SET         :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, welcher in den CONFIG-Modus versetzt werden soll. Für den lokalen TwinCAT-Rechner kann auch ein Leerstring angegeben werden.

SET: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

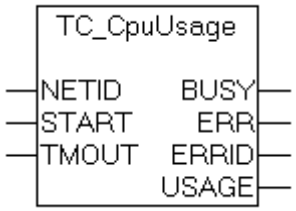
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.50 TC_CpuUsage



Mit dem Funktionsbaustein "TC_CpuUsage" kann die aktuelle CPU-Auslastung eines TwinCAT-Systems ermittelt werden. Die Funktion entspricht der Anzeige der CPU-Auslastung im TwinCAT-System Menü unter den Echtzeiteinstellungen. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen CPU-Auslastung ermittelt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

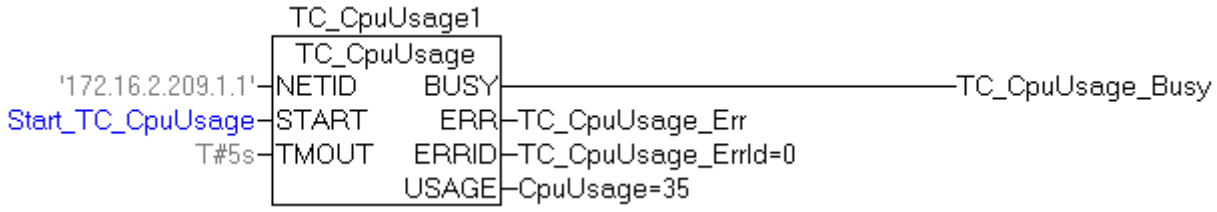
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

USAGE: Die aktuelle CPU-Auslastung eines TwinCAT-Systems in %.

Beispiel für einen Aufruf in FUP

```
TC_CpuUsage1          : TC_CpuUsage;
Start_TC_CpuUsage     : BOOL;
TC_CpuUsage_Busy      : BOOL;
TC_CpuUsage_Err       : BOOL;
TC_CpuUsage_ErrId     : UDINT;
CpuUsage              : UDINT;
```

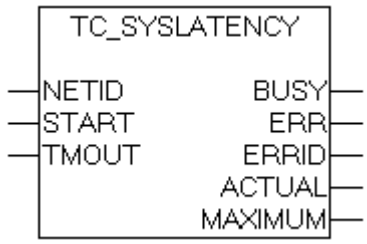


In dem Beispiel benutzt das TwinCAT-System 35% von der gesamten zur Verfügung stehenden CPU-Rechenzeit.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.51 TC_SysLatency



Mit dem Funktionsbaustein "TC_SysLatency" kann die aktuelle und maximale Latenzzeit eines TwinCAT-Systems ermittelt werden. Die Funktion entspricht der Anzeige der TwinCAT-Latenzzeit im TwinCAT-System Menü unter den Echtzeiteinstellungen. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    NETID       :T_AmsNetId;
    START       :BOOL;
    TMOUT       :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen Latenzzeit ermittelt werden soll. Für den lokalen Rechner kann auch ein Leerstring angegeben werden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

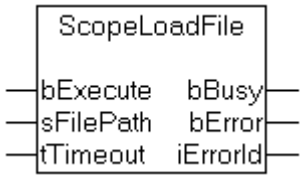
ACTUAL: Die aktuelle Latenzzeit eines TwinCAT-Systems in µs.

MAXIMUM: Die maximale Latenzzeit eines TwinCAT-Systems in µs (Maximale Latenzzeit, die seit dem letzten TwinCAT System-Start aufgetreten ist).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0, Build > 732	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.52 ScopeLoadFile



Mit dem Funktionsbaustein "ScopeLoadFile" kann ein Scope View Configuration Project (*.scp) in dasTwinCAT Scope View geladen werden. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

sFilePath:: Der gesamte Pfad mit dem Namen der Datei die in das Scope View geladen werden soll (z.B.: 'C:\TwinCAT\Scope\Achse1.scp').

tTimeout:: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

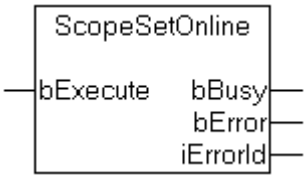
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.53 ScopeSetOnline



Mit dem Funktionsbaustein "ScopeSetOnline" wird das Scope View in den Online-Status geschaltet. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

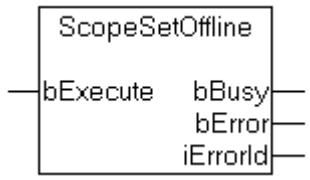
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.54 ScopeSetOffline



Mit dem Funktionsbaustein "ScopeSetOffline" wird das Scope View in den Offline-Zustand geschaltet. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

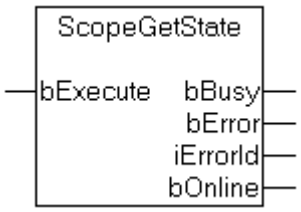
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.55 ScopeGetState



Mit dem Funktionsbaustein "ScopeGetState" kann der Zustand des Scope Views ermittelt werden. Befindet sich das Scope View im Online-Zustand, dann wird der bOnline-Ausgang gesetzt. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    iErrorId    :UDINT;
    bOnline     :BOOL;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

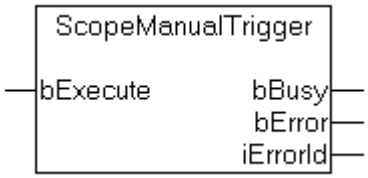
iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

bOnline: Ist dieser Ausgang gesetzt, dann befindet sich das Scope View im Online-Zustand, sonst im Offline-Zustand.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.56 ScopeManualTrigger



Mit dem Funktionsbaustein "ScopeManualTrigger" kann das Scope View manuell getriggert werden. Diese Funktion kann zur Zeit nur genutzt werden, wenn nur ein View (z.B. Scope View 1) in der Applikation aktiv ist. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

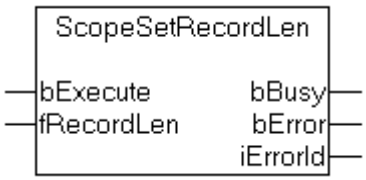
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.57 ScopeSetRecordLen



Mit dem Funktionsbaustein "ScopeSetRecordLen" kann die Aufnahme-Zeit des Scope View konfiguriert werden. Diese Funktion kann zur Zeit nur genutzt werden, wenn nur ein View (z.B. Scope View 1) in der Applikation aktiv ist. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    fRecordLen      :LREAL;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

fRecordLen: Die zu konfigurierende Aufnahme-Zeit für das Scope View in Sekunden.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

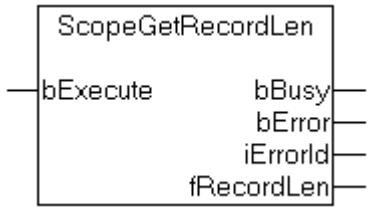
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.58 ScopeGetRecordLen



Mit dem Funktionsbaustein "ScopeGetRecordLen" kann die aktuell im TwinCAT Scope View konfigurierte Aufnahme-Zeit für das Scope View ermittelt werden. Diese Funktion kann zur Zeit nur genutzt werden, wenn nur ein View (z.B. Scope View 1) in der Applikation aktiv ist. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    iErrorId    : UDINT;
    fRecordLen  : LREAL;
END_VAR
```

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

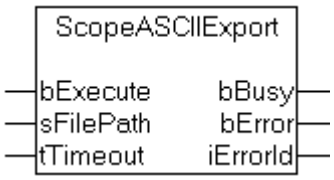
iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

fRecordLen: Die aktuell konfigurierte Aufnahme-Zeit für das Scope View in Sekunden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.59 ScopeASCIIExport



Mit dem Funktionsbaustein "ScopeASCIIExport" kann das Scope View in eine ASCII-Datei exportiert werden. Diese Funktion kann zur Zeit nur genutzt werden, wenn nur ein View (z.B. Scope View 1) in der Applikation aktiv ist. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

sFilePath:: Der gesamte Pfad mit dem Namen der Datei in die exportiert werden soll (z.B.: 'C:\TwinCAT\Scope\Achse1.dat').

tTimeout:: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

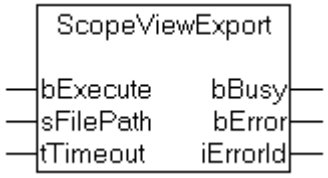
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.60 ScopeViewExport



Mit dem Funktionsbaustein "ScopeViewExport" kann ein Scope View in eine Datei (*.svw) exportiert werden. Diese Funktion kann zur Zeit nur genutzt werden, wenn nur ein View (z.B. Scope View 1) in der Applikation aktiv ist. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

sFilePath:: Der gesamte Pfad mit dem Namen der Datei in die exportiert werden soll (z.B.: 'C:\TwinCAT\Scope\Achse1.svw').

tTimeout:: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

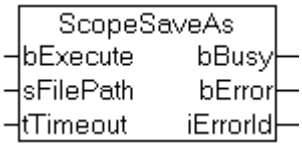
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.61 ScopeSaveAs



Mit dem Funktionsbaustein "ScopeSaveAs" kann ein Scope View Configuration Project (*.scp) unter einem bestimmten Dateinamen abgespeichert werden. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;
    sFilePath       :STRING;
    tTimeout        :TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

sFilePath:: Der gesamte Pfad mit dem Namen der Datei (z.B.: 'C:\TwinCAT\Scope\Achse1.scp').

tTimeout:: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

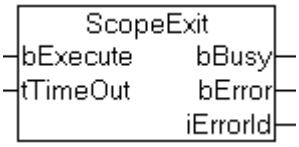
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0 Build > 740 TwinCAT v2.9.0 Build > 925	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.62 ScopeExit



Mit dem Funktionsbaustein "ScopeExit" kann eine Instanz des TwinCAT Scope View beendet werden. Die Funktion entspricht dem Datei -> Beenden-Kommando aus dem Scope View Menü. Intern wird eine Instanz des ADSWRITE-Funktionsbausteins aufgerufen.

VAR_INPUT

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

bExecute:: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

tTimeout:: Maximale Zeit, die bei der Ausführung des Funktionsbausteins nicht überschritten werden darf.

VAR_OUTPUT

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

bBusy: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

iErrorId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

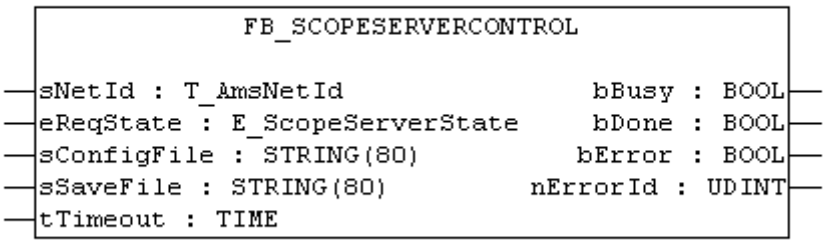
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0 Build > 740 TwinCAT v2.9.0 Build > 925	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.63 FB_ScopeServerControl

[Dies ist eine vorläufige Dokumentation und unterliegt noch Änderungen]

- Ab Bibliotheks-Version 2.0.52
- Erfordert die Installation des TwinCAT Scope Servers, der mit dem TwinCAT Scope View 2 geliefert wird.



Der Funktionsbaustein "FB_ScopeServerControl" ermöglicht der SPS Daten zu sammeln, die dann später mit TwinCAT Scope View 2 angezeigt werden sollen.

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: Hier kann ein String angegeben werden der die Netzwerkadresse des TwinCAT Zielsystems beinhaltet. Für den lokalen Computer kann dieser String auch leer sein.

eReqState:: Angeforderter Scope Server Status [▶ 209].

sConfigFile:: Vollständiger Pfad mit dem Namen der Konfigurations-Datei (z.B.: 'C:\TwinCAT\TwinCATScopeView2\First.sv2').

sSaveFile: Vollständiger Pfad mit dem Namen der Daten-Datei (z.B: 'C:\TwinCAT\TwinCATScopeView2\First.svd').

tTimeout:: Maximal zulässige Zeit für die internen ADS Kommandos

VAR_OUTPUT

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

bBusy:Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

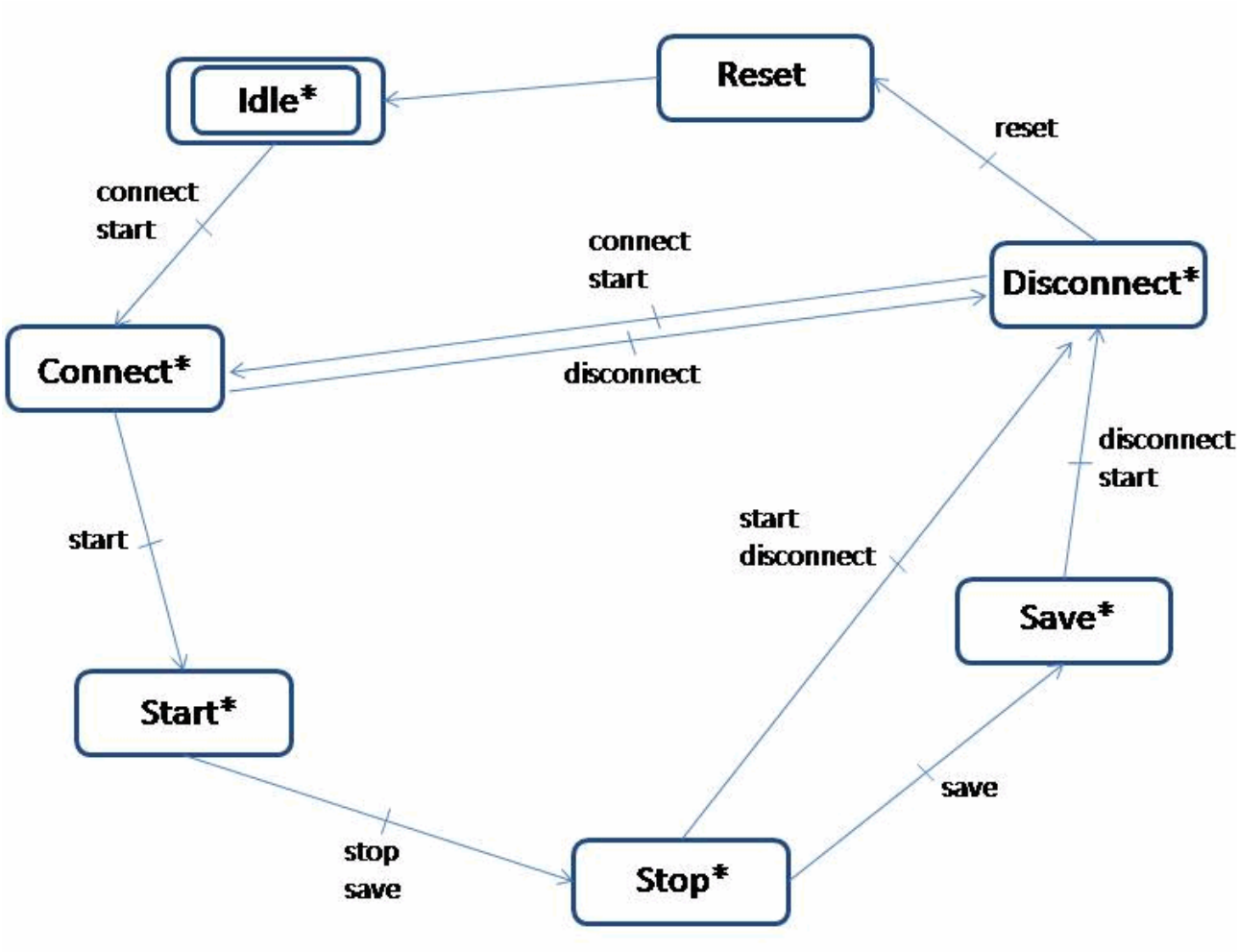
bDone: Wird gesetzt wenn der angeforderte Status aktiviert wurde.

bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der *bBusy*-Ausgang zurückgesetzt wurde.

nErrorId: Zeigt die Fehlernummer, wenn der Fehlerausgang gesetzt ist. Grundsätzlich kann dies eine ADS-Fehlernummer oder eine spezifischer Fehler-Code [▶ 251] dieser Bibliothek sein.

Es ist nur ein Zielsystem für die Konfigurationsdatei (*.sv2) zugelassen.

Status-Diagramm:



*) eine Statusänderung zurückzusetzen ist jederzeit möglich

Dieses Status-Diagramm zeigt die möglichen Übergänge für eReqState. Falls ein anderer Statusübergang angefordert wird, wird bError gesetzt.

TYPE 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:

Deklarationsteil:

```
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
```

Implementierung des 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);
```

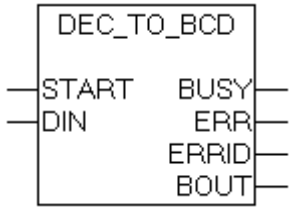
Dieses Beispiel soll zeigen, wie eine Langzeitaufzeichnung mit dem Scope Server durchgeführt werden kann.

Dafür wird eine existierende Konfiguration (Test.sv2) geladen. In diesem Beispiel wurde Test.sv2 gespeichert, um im Ring-Buffer zu laufen. So wird die Datenaufzeichnung nicht enden bis sie von FB_ScopeServerControl angesteuert wird. Falls ein internes Trigger- Ereignis (dies kann ein Fehlerereignis sein) auftritt, wird ein Timer gestartet und 10 Sekunden später werden die Daten in Test.svd gespeichert. Auf diese Weise beinhaltet die Daten-Datei Informationen vor und nach dem Trigger-Ereignis.

Voraussetzungen

Entwicklungsumgebung	Art des Zielsystems	zu inkludierende PLC-Bibliotheken
TwinCAT v2.10.0 Build >= 1329	PC or CX (x86,ARM)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib werden automatisch inkludiert)

4.64 DEC_TO_BCD



Mit dem Funktionsbaustein "DEC_TO_BCD" können Dezimal-Zahlen in BCD-Format konvertiert werden. Die zu konvertierende Zahl wird auf Zulässigkeit der Werte überprüft.

VAR_INPUT

```
VAR_INPUT
    START          : BOOL;
    DIN            : BYTE;
END_VAR
```

START: Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

DIN: Die zu konvertierende Dezimalzahl.

VAR_OUTPUT

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

BUSY: Beim Starten der Konvertierung wird dieser Ausgang gesetzt und bleibt gesetzt, bis die Konvertierung abgeschlossen wurde. Nachdem der BUSY - Ausgang zurückgesetzt wurde, steht die BCD-Zahl am BOUT-Ausgang zur Verfügung.

ERR: Beim Fehler wird die Variable auf TRUE gesetzt.

ERRID: Fehlercode.

BOUT: Beim Erfolg steht an diesem Ausgang die konvertierte Variable im BCD-Format.

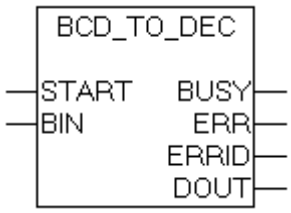
Fehlercodes:

Fehlercode	Fehlerbeschreibung
0	kein Fehler
0x00FF	Unzulässiger Dezimalwert der zu konvertierenden Variablen

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.65 BCD_TO_DEC



Mit dem Funktionsbaustein "BCD_TO_DEC" können BCD-Zahlen in Dezimal - Format konvertiert werden. Die zu konvertierende BCD-Zahl wird auf Zulässigkeit der Werte überprüft.

VAR_INPUT

```
VAR_INPUT
    START          : BOOL;
    BIN            : BYTE;
END_VAR
```

START: Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

BIN: Die zu konvertierende BCD-Zahl.

VAR_OUTPUT

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

BUSY: Beim Starten der Konvertierung wird dieser Ausgang gesetzt und bleibt gesetzt, bis die Konvertierung abgeschlossen wurde. Nachdem der BUSY - Ausgang zurückgesetzt wurde, steht der Dezimalwert am DOUT-Ausgang zur Verfügung.

ERR: Beim Fehler wird die Variable auf TRUE gesetzt.

ERRID: Fehlercode.

DOUT: Beim Erfolg steht an diesem Ausgang die konvertierte Variable im Dezimal - Format.

Fehlercodes:

Fehlercode	Fehlerbeschreibung
0	kein Fehler
0x000F	Unzulässiger Wert im Low-Nibble der BCD-Zahl
0x00F0	Unzulässiger Wert im High-Nibble der BCD-Zahl

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.66 RTC

Mit dem Funktionsbaustein "RTC" (Real Time Clock) kann eine interne Software-Uhr in der TwinCAT SPS realisiert werden. Die Uhr muss mit einem Anfangsdatum und einer Uhrzeit initialisiert werden. Nach der Initialisierung wird die Uhrzeit und das Datum mit jedem Aufruf des Funktionsbausteins aktualisiert. Um die aktuelle Uhrzeit und das Datum zu berechnen, wird ein Systemtakt der CPU benutzt. Damit die aktuelle Zeit berechnet werden kann, sollte der Funktionsbaustein in jedem Zyklus der SPS ein Mal aufgerufen werden. Am Ausgang des Funktionsbausteines steht das aktuelle Datum und Uhrzeit in dem gängigen DATE_AND_TIME (DT) Format zur Verfügung. In einem SPS-Programm können mehrere Instanzen von dem RTC-Funktionsbaustein erzeugt werden.

Bedingt durch die Systemeigenschaften weicht die RTC-Zeit von einer Referenzzeit ab. Die Abweichung hängt von der Zykluszeit der SPS, dem Wert des System-Basis-Ticks und der verwendeten Hardware. Um größere Abweichungen zu vermeiden, muss die RTC-Instanz zyklisch (z. B. mit einer Funkuhr oder mit der lokalen Windows-Systemzeit) synchronisiert werden. Die lokale Windows-Systemzeit können Sie wiederum mit der Hilfe des SNTP-Protokolls mit einer Referenzzeit synchronisieren.

VAR_INPUT

```
VAR_INPUT
    EN   : BOOL;
    PDT  : DATE_AND_TIME;
END_VAR
```

EN: Bei einer steigenden Flanke an diesem Eingang wird der Funktionsbaustein mit einer vorgegebenen Uhrzeit und Datum neu initialisiert.
PDT: (Preset Date and Time) Die Initialisierungswerte für das Datum und Uhrzeit des Funktionsbausteins. Bei einer steigenden Flanke an dem EN-Eingang wird dieser Wert von dem Funktionsbaustein übernommen.

VAR_OUTPUT

```
VAR_OUTPUT
    Q    : BOOL;
    CDT  : DATE_AND_TIME;
END_VAR
```

Q: Wurde der Funktionsbaustein mindestens ein Mal initialisiert, wird dieser Ausgang gesetzt. Ist dieser Ausgang gesetzt, dann sind die Werte für das Datum und Uhrzeit am PDT-Ausgang gültig.
CDT: (Current Date and Time) Aktuelles Datum und Uhrzeit von der RTC-Instanz. Der CDT-Ausgang wird nur dann aktualisiert, wenn der Funktionsbaustein aufgerufen wurde. Daher sollten die Instanzen des Funktionsbausteines ein Mal in jedem Zyklus der SPS aufgerufen werden.

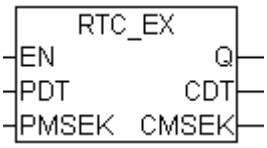
Beispiel:

Siehe: Beispiel: Software-Uhren (RTC, RTC_EX, RTC_EX2) [▶ 273].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1030 und höher TwinCAT v2.10.0 Build > 1232 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.67 RTC_EX



Mit dem Funktionsbaustein "RTC_EX" (Extended Real Time Clock) kann eine interne Software-Uhr in der TwinCAT SPS realisiert werden. Die Uhr muss mit einem Anfangsdatum und einer Uhrzeit initialisiert werden. Nach der Initialisierung wird die Uhrzeit und das Datum mit jedem Aufruf des Funktionsbausteins aktualisiert. Um die aktuelle Uhrzeit und das Datum zu berechnen, wird ein Systemtakt der CPU benutzt. Damit die aktuelle Zeit berechnet werden kann, sollte der Funktionsbaustein in jedem Zyklus der SPS ein Mal aufgerufen werden. Am Ausgang des Funktionsbausteines steht das aktuelle Datum und die Uhrzeit in dem gängigen DATE_AND_TIME (DT) Format zur Verfügung. Im Gegensatz zu dem RTC [▶ 213]-Funktionsbaustein hat RTC_EX eine Millisekunden-Genauigkeit. In einem SPS-Programm können mehrere Instanzen von dem RTC_EX-Funktionsbaustein erzeugt werden.

Bedingt durch die Systemeigenschaften weicht die RTC_EX-Zeit von einer Referenzzeit ab. Die Abweichung hängt ab von der Zykluszeit der SPS, dem Wert des System-Basis-Ticks und der verwendeten Hardware. Um größere Abweichungen zu vermeiden muss die RTC_EX-Instanz zyklisch (z. B. mit einer Funkuhr oder mit der lokalen Windows-Systemzeit) synchronisiert werden. Die lokale Windows-Systemzeit können Sie wiederum mit der Hilfe des SNTP-Protokolls mit einer Referenzzeit synchronisieren.

VAR_INPUT

```
VAR_INPUT
   EN     : BOOL;
   PDT    : DATE_AND_TIME;
   PMSEK  : DWORD;
END_VAR
```

EN: Bei einer steigenden Flanke an diesem Eingang wird der RTC_EX-Funktionsbaustein mit vorgegebener Uhrzeit, Datum und Millisekunden neu initialisiert.
PDT: (Preset Date and Time) Die Initialisierungswerte für Datum und Uhrzeit des Funktionsbausteins. Bei einer steigenden Flanke an dem EN-Eingang wird dieser Wert von dem Funktionsbaustein übernommen.
PMSEK: (Preset Milliseconds) Der Initialisierungswert für die Millisekunden. Bei einer steigenden Flanke an dem EN-Eingang wird dieser Wert von dem Funktionsbaustein übernommen.

VAR_OUTPUT

```
VAR_OUTPUT
   Q      : BOOL;
   CDT    : DATE_AND_TIME;
   CMSEK  : DWORD;
END_VAR
```

Q: Wurde der Funktionsbaustein mindestens ein Mal initialisiert, wird dieser Ausgang gesetzt. Ist dieser Ausgang gesetzt, dann sind die Werte für Datum, Uhrzeit und Millisekunden am PDT-Ausgang und CMSEK-Ausgang gültig.
CDT: (Current Date and Time) Aktuelles Datum und Uhrzeit von der RTC_EX-Instanz. Der CDT-Ausgang wird nur dann aktualisiert, wenn der Funktionsbaustein aufgerufen wurde. Daher sollten die Instanzen des Funktionsbausteines ein Mal in jedem Zyklus der SPS aufgerufen werden.
CMSEK: (Current Milliseconds) Der Millisekunden-Ausgang.

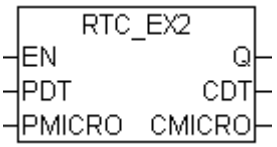
Beispiel:

Siehe: Beispiel: Software-Uhren (RTC, RTC_EX, RTC_EX2) [▶ 273].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1030 und höher TwinCAT v2.10.0 Build > 1232 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.68 RTC_EX2



Mit dem Funktionsbaustein "RTC_EX2" (Extended Real Time Clock) kann eine interne Software-Uhr in der TwinCAT SPS realisiert werden. Die Uhr muss mit einem Anfangsdatum und einer Uhrzeit initialisiert werden. Nach der Initialisierung wird die Uhrzeit und das Datum mit jedem Aufruf des Funktionsbausteins aktualisiert. Um die aktuelle Uhrzeit und das Datum zu berechnen, wird ein Systemtakt der CPU benutzt. Damit die aktuelle Zeit berechnet werden kann, sollte der Funktionsbaustein in jedem Zyklus der SPS ein Mal aufgerufen werden. Am Ausgang des Funktionsbausteines steht das aktuelle Datum und Uhrzeit in dem Windows-Systemzeitformat zur Verfügung. Im Gegensatz zu dem RTC [▶ 213]-Funktionsbaustein hat RTC_EX2 eine Mikrosekunden-Genauigkeit. In einem SPS-Programm können mehrere Instanzen von dem RTC_EX2-Funktionsbaustein erzeugt werden.

> **i** Bedingt durch die Systemeigenschaften weicht die RTC_EX2-Zeit von einer Referenzzeit ab. Die Abweichung ist abhängig von der Zykluszeit der SPS, dem Wert des System-Basis-Ticks und der verwendeten Hardware. Um größere Abweichungen zu vermeiden muss die RTC_EX2-Instanz zyklisch (z. B. mit einer Funkuhr oder mit der lokalen Windows-Systemzeit) synchronisiert werden. Die lokale Windows-Systemzeit können Sie wiederum mit der Hilfe des SNTP-Protokolls mit einer Referenzzeit synchronisieren.

VAR_INPUT

```
VAR_INPUT
   EN       : BOOL;
   PDT      : TIMESTRUCT;
   PMICRO   : DWORD;
END_VAR
```

EN: Bei einer steigenden Flanke an diesem Eingang wird der RTC_EX2-Funktionsbaustein mit vorgegebener Uhrzeit, Datum und Millisekunden neu initialisiert.
PDT: (Preset Date and Time) Die Initialisierungswerte für Datum und Uhrzeit des Funktionsbausteins. Bei einer steigenden Flanke an dem EN-Eingang wird dieser Wert von dem Funktionsbaustein übernommen.
PMICRO: (Preset Microseconds) Der Initialisierungswert für die Microsekunden. Bei einer steigenden Flanke an dem EN-Eingang wird dieser Wert von dem Funktionsbaustein übernommen.

VAR_OUTPUT

```
VAR_OUTPUT
   Q        : BOOL;
   CDT      : TIMESTRUCT;
   CMICRO   : DWORD;
END_VAR
```

Q: Wurde der Funktionsbaustein mindestens ein Mal initialisiert, wird dieser Ausgang gesetzt. Ist dieser Ausgang gesetzt, dann sind die Werte für Datum, Uhrzeit und Millisekunden am PDT-Ausgang und CMICRO-Ausgang gültig.
CDT: (Current Date and Time) Aktuelles Datum und Uhrzeit von der RTC_EX2-Instanz. Der CDT-Ausgang

wird nur dann aktualisiert, wenn der Funktionsbaustein aufgerufen wurde. Daher sollten die Instanzen des Funktionsbausteines ein Mal in jedem Zyklus der SPS aufgerufen werden.
CMICRO: (Current Microseconds) Der Microsekunden-Ausgang.

Beispiel:

Siehe: Beispiel: Software-Uhren (RTC, RTC_EX, RTC_EX2) [▶ 273].

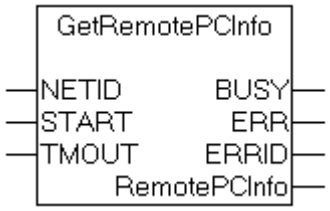
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build > 1030 und höher TwinCAT v2.10.0 Build > 1232 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

TIMESTRUCT [▶ 231]

4.69 GetRemotePCInfo



Mit dem Funktionsbaustein "GetRemotePCInfo" können Informationen über konfigurierte Remote-PCs im TwinCAT - Router ausgelesen werden. Nach einer erfolgreichen Ausführung sind in der Struktur "RemotePCInfo" die NetIds und Namen der Remote-PCs als Strings abgelegt, in der Reihenfolge, wie sie im TwinCAT - Router abgelegt wurden. Intern wird eine Instanz des ADSREAD-Funktionsbausteins aufgerufen. Mit dem Funktionsbaustein können Router-Informationen des lokalen oder eines Remote TwinCAT-Systems ausgelesen werden.

VAR_INPUT

```
VAR_INPUT
    NETID       : T_AmsNetId;
    START       : BOOL;
    TMOUT       : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, dessen Router-Informationen über konfigurierte Remote-PCs ausgelesen werden soll. Sollen die Remote-PCs des lokalen TwinCAT-Systems ermittelt werden, kann auch ein Leerstring angegeben werden.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
```

```
    ERRID        : UDINT;
    RemotePCInfo : REMOTEPCINFOSTRUCT;
END_VAR
```

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

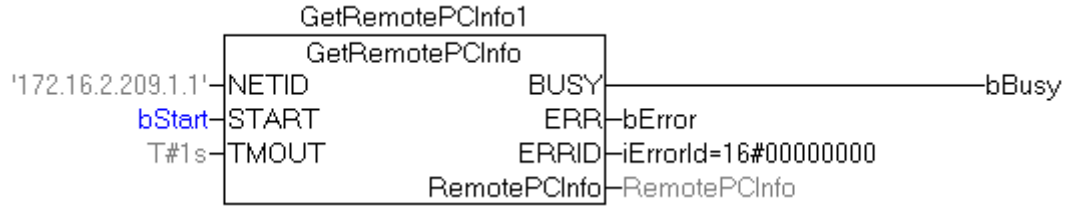
ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

RemotePCInfo: Struktur [▶ 232] mit Informationen zu den konfigurierten Remote-PCs.

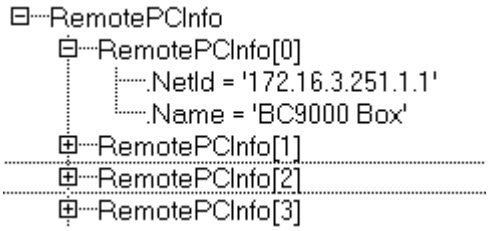
Beispiel für einen Aufruf in FUP:

```
PROGRAM MAIN
VAR
    GetRemotePCInfo1    : GetRemotePCInfo;
    RemotePCInfo        : REMOTEPCINFOSTRUCT;

    bBusy       : BOOL;
    bError      : BOOL;
    iErrorId    : UDINT;
    bStart      : BOOL;
END_VAR
```



Online-Ansicht:

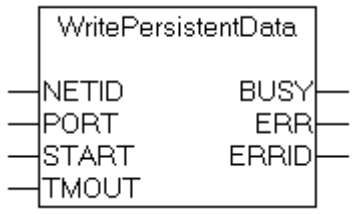
Die NetIds und Namen der konfigurierten Remote-PCs.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.70 WritePersistentData



Wurden in einem Laufzeitsystem der SPS persistente Variablen definiert, dann werden deren aktuelle Werte beim TwinCAT-System-Stopp (nach dem letztem Zyklus der SPS) in eine Datei im ...TwinCAT\Boot-Ordner gespeichert. Für jedes konfigurierte Laufzeitsystem wird eine Datei angelegt. Beim nächsten Systemstart wird diese Datei eingelesen und die persistenten Variablen im Laufzeitsystem mit den Werten aus der Datei initialisiert. Mit dem Funktionsbaustein "WritePersistentData" kann das Speichern der persistenten Daten aus dem SPS-Programm ausgelöst werden. Der PORT-Parameter bestimmt das Laufzeitsystem dessen persistente Daten gespeichert werden sollen. Siehe auch: Schreiben der pers. Daten: Systemverhalten [▶ 278]. Intern wird eine Instanz des ADSWRTCTRL-Funktionsbausteins aufgerufen.

VAR_INPUT

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

NETID: Hier kann die AmsNetId des TwinCAT-Rechners angegeben werden, auf dem das ADS-Kommando ausgeführt werden soll. Für den lokalen Rechner, kann auch ein Leerstring angegeben werden.

PORT: Beinhaltet die ADS-Portnummer des SPS-Laufzeitsystems dessen persistente Daten gespeichert werden sollen. Das Erste SPS-Laufzeitsystem hat z.B. die Portnummer 801 und das Zweite die Portnummer 811.

START: Über eine positive Flanke an diesem Eingang wird der Baustein aktiviert.

TMOUT: Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

VAR_OUTPUT

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

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt.

ERR: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der BUSY-Ausgang zurückgesetzt wurde.

ERRID: Liefert bei einem gesetzten ERR-Ausgang die ADS-Fehlernummer.

SPS-Beispielprogramm 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
```

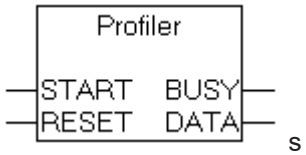
```
fbR_Trig( CLK:=bStart );

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;
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

4.71 Profiler

Diese Funktionalität ist in der SPS unter Windows CE nicht verfügbar!



Mit dem Funktionsbaustein "Profiler" können Messungen der Ausführungszeit vom SPS-Code durchgeführt werden. Intern wird eine Instanz des GETCPUACCOUNT-Funktionsbausteines aufgerufen. Über eine steigende Flanke an dem START-Eingang wird der Messvorgang gestartet und bei einer fallenden Flanke gestoppt. Die Messungen werden intern ausgewertet und stehen am DATA-Ausgang in einer Struktur vom Typ PROFILERSTRUCT [▶ 232] für weitere Verarbeitung zur Verfügung. Neben der aktuellen, minimalen und maximalen Ausführungszeit wird von dem Funktionsbaustein die mittlere Ausführungszeit der letzten 10 Messungen berechnet. Die Anzahl der gemittelten Messwerte kann über die globale Variable MAX_AVERAGE_MEASURES [▶ 250] auf 2 bis 100 Messwerte konfiguriert werden. Die gemessenen Zeiten werden in Mikrosekunden ausgegeben. Die Ausgangsvariable DATA.MasureCycle [▶ 232] gibt Auskunft über die gesamte Anzahl der bereits durchgeführten Messungen. Um die Ausführungszeit von einem bestimmten Programmabschnitt der SPS zu messen, muss der Messvorgang über eine steigende START-Flanke am

Anfang des zu messenden Programmabschnitts gestartet und am Ende über eine fallende START-Flanke gestoppt werden. Alle Werte am DATA-Ausgang können zurückgesetzt werden wenn gleichzeitig mit der steigenden Flanke am START auch eine steigende Flanke am RESET-Eingang erzeugt wird. Die alten Messwerte werden so zu Beginn einer neuen Messung zurückgesetzt und aus den folgenden Aufrufen des Funktionsbausteins neu berechnet.

Die ermittelten Zeiten können von den tatsächlichen Werten abweichen, da bereits für die Aufrufe des GETCPUACCOUNT-Funktionsbausteins einige Zeit benötigt wird. Diese Zeit ist rechnerabhängig und ist in den ermittelten Zeiten enthalten.

VAR_INPUT

```
VAR_INPUT
    START          :BOOL;
    RESET          :BOOL;
END_VAR
```

START: Über eine positive Flanke an diesem Eingang wird die Messung der Ausführungszeit gestartet. Über eine negative Flanke an diesem Eingang wird die Messung gestoppt und die aktuelle, minimale, maximale und mittlere Ausführungszeit neu berechnet. Die Variable DATA.MeasureCycle [▶ 232] wird dabei inkrementiert.

RESET: Wenn gleichzeitig mit einer steigenden Flanke am START-Eingang auch eine steigende Flanke an diesem Eingang erzeugt wurde dann werden alle Variablen am DATA-Ausgang zurückgesetzt. Die alten Werte für die aktuelle, minimale, maximale und mittlere Ausführungszeit werden dabei zurückgesetzt und für die nächsten Messungen neu berechnet.

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY           :BOOL;
    DATA           :PROFILERSTRUCT;
END_VAR
```

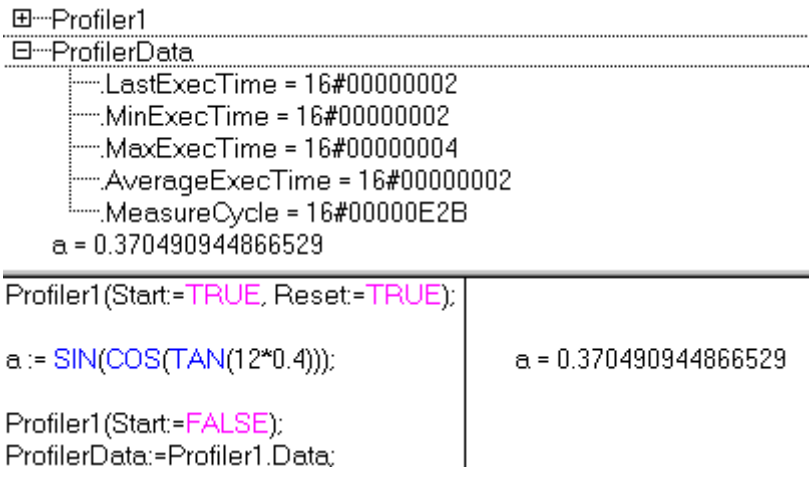
BUSY: Beim Starten des Messvorgangs wird dieser Ausgang gesetzt und bleibt gesetzt, bis die Zeitmessung abgeschlossen wurde. Nachdem der BUSY-Ausgang zurückgesetzt wurde, stehen die aktuellen Zeiten am DATA-Ausgang zur Verfügung.

DATA: Struktur vom Typ PROFILERSTRUCT [▶ 232] mit den gemessenen Zeiten in [µs].

Beispiel für einen Aufruf in ST:

```
PROGRAM ProfilerTest_ST
VAR
    Profiler1      :PROFILER;
    ProfilerData   :PROFILERSTRUCT;
    a          :LREAL;
END_VAR
```

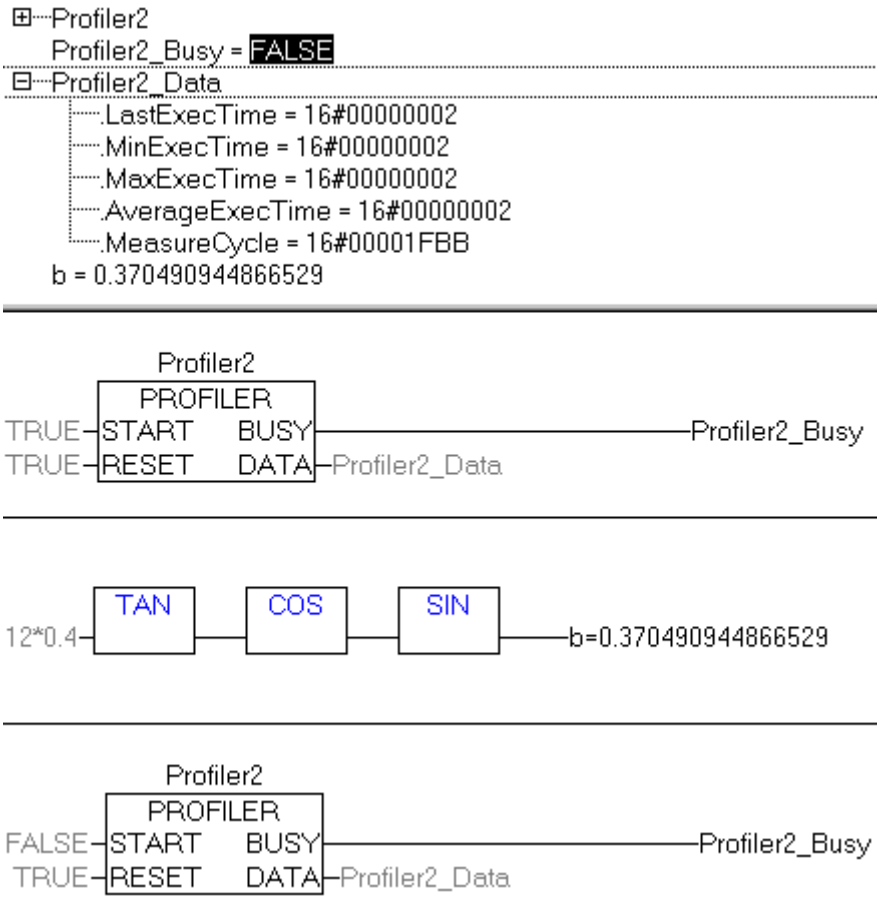
Online Ansicht der gemessenen Zeiten:



Beispiel für einen Aufruf in FUP:

```
PROGRAM ProfilerTest_FUP
VAR
    Profiler2     :PROFILER;
    Profiler2_Busy    :BOOL;
    Profiler2_Data    :PROFILERSTRUCT;
    b         :LREAL;
END_VAR
```

Online Ansicht der gemessenen Zeiten:



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
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

4.72 DCF77_TIME

Mit dem Funktionsbaustein "DCF77_TIME" kann das DCF-77 Funkuhr-Signal dekodiert werden.

Dieser Funktionsbaustein wird von dem neuen "DCF77_TIME_EX [▶ 225]"-Funktionsbaustein ersetzt.

Über eine steigende Flanke am RUN-Eingang wird die Dekodierung gestartet fortgesetzt, solange der RUN-Eingang gesetzt ist. Der Funktionsbaustein braucht im schlechtesten Fall max. 1 Minute um sich zu synchronisieren und eine weitere Minute um Daten für die nächste Minute zu dekodieren. Während dieser Zeit wird auf die fehlende 59. Sekundenmarke gewartet. Intern findet bei dem Funktionsbaustein eine Abtastung des DCF-77 Signals statt. Um die Flanken fehlerfrei abtasten zu können, sollte der Funktionsbaustein in jedem Zyklus der SPS einmal aufgerufen werden. Bei einer Zykluszeit <= 25ms können zufriedenstellende Ergebnisse erzielt werden. Bei einem fehlenden oder fehlerhaften DCF-77 Signal wird der ERR-Ausgang auf TRUE gesetzt und ein entsprechender Fehlercode am ERRID-Ausgang gesetzt. Bei dem nächsten fehlerfreien Empfang werden die ERR- und ERRID-Ausgänge zurückgesetzt. Manche der Empfänger liefern ein invertiertes DCF-77 Signal. In so einem Fall muss das Signal zuerst invertiert werden und dann an den DCF_PULSE-Eingang geleitet werden. Bei einem fehlerfreien Betrieb wird die aktuelle Zeit im Minutentakt am CDT-Ausgang aktualisiert. Dabei wird für einen Zyklus der SPS bei der Nullten Sekunde der READY-Ausgang auf TRUE gesetzt. In dieser Zeit ist die DCF-77 Zeit am CDT-Ausgang gültig und kann im SPS-Programm ausgewertet werden. Der READY-Ausgang wird nur dann gesetzt, wenn kein Fehler in den Daten für die kommende Minute erkannt wurde. Die Fehlererkennung wird mit Hilfe der übertragenen Paritätsbits durchgeführt. Bei schlechten Empfangsverhältnissen kann eine zu 100% fehlerfreie Erkennung nicht gewährleistet werden. D.h. bei zwei fehlerhaften (invertierten) Bits kann der Funktionsbaustein keinen Fehler erkennen und setzt den READY-Ausgang ebenfalls auf TRUE. Um eine zuverlässige Zeitinformation zu erhalten müssen zusätzliche Sicherungsmechanismen implementiert werden z.B. die Auswertung der Redundanz der Zeitinformation in aufeinanderfolgenden Minuten.

Ab TwinCAT v2.10 Build > 1340 und TwinCAT v2.11 Build > 1542 wurde in dem DCF77_TIME-Funktionsbaustein eine einfache Plausibilitätsprüfung von zwei aufeinanderfolgenden Telegrammen implementiert. Diese Funktionalität kann über eine globale boolesche Variable für alle Instanzen des DCF77_TIME-Bausteins aktiviert werden. Bei der Aktivierung der Plausibilitätsprüfung verlängert sich die erste Synchronisierung um eine weitere Minute auf max. 3 Minuten.

```
GLOBAL_DCF77_SEQUENCE_CHECK : BOOL := FALSE;
(* TRUE = Enable plausibility check (two telegrams are checked), FALSE = Disable check *)
```

Die auftretenden Fehler während des Empfangs werden von dem Funktionsbaustein registriert. Der ERRCNT-Ausgang ist ein Fehlerzähler. Der Zähler gibt Auskunft über die Anzahl der aufgetretenen Fehler seit dem letzten fehlerfreien Empfang. Bei dem nächsten fehlerfreien Empfang wird dieser Zähler zurückgesetzt.

Zeitcode

Während jeder Minute werden die Nummern von Jahr, Monat, Tag, Wochentag, Stunde, Minute BCD-codiert durch Impulsmodulation der Sekundenmarken übertragen. Die übertragenen Informationen gelten jeweils für die nachfolgende Minute. Jede Sekunde wird eine der Sekundenmarken übertragen. Dabei entspricht eine

Sekundenmarke mit der Dauer von 0.1s einer binären Null und eine mit der Dauer von 0.2s einer binären Eins. Die Informationen werden mit 3 Prüfbits ergänzt. Bei der 59. Sekunde fehlt die Sekundenmarke und diese "Lücke" kann zur Synchronisation des Empfängers benutzt werden.

Ab TwinCAT v2.10 Build > 1340 und TwinCAT v2.11 Build > 1542 kann auch über eine globale Variable die Länge des kurzen und langen Impulssignals konfiguriert werden. Bei einem gestörten Signal sind die Impulsbreiten immer kleiner. In der Empfängerspezifikation sind meistens Angaben über minimale und maximale Impulsbreiten der beiden logischen Signale zu finden, wobei breitere Impulse bei höheren Feldstärken zu erwarten sind und schmale Impulse bei niedrigen Feldstärken oder entsprechenden Störverhältnissen. In der Nähe des Senders (wo die Feldstärke sehr groß ist) können ebenfalls Probleme entstehen wenn die Impulsbreite der logischen Null zu groß wird. Deshalb wird abhängig von der Spezifikation des Empfängers eine feste Grenze zur Unterscheidung zwischen Null und Eins festgelegt. **Prüfen Sie deshalb die Spezifikation des eingesetzten Empfängers und konfigurieren entsprechend die Impulslänge.**

```
GLOBAL_DCF77_PULSE_SPLIT : TIME := T#140ms;(* 0 == pulse < 140ms, 1 == pulse > 140 *)
```

Z.B.: in der Spezifikation von Atmel T4227 (Time Code Receiver) wird folgende Pulslänge angegeben:
100ms Pulse (Null): Min: 70ms, Typical: 95ms, Max: **130ms**
200ms Pulse (Eins): Min **170ms**, Typical 195ms, Max 235ms
Für diesen IC wäre ein Grenzwert von 150ms optimal = 130 + ((170ms - 130ms) / 2).

Wenn der konfigurierte Grenzwert für die Impulslänge einen zu kleinen Wert hat, dann werden kurze Impulse als lange erkannt. Umgekehrt gilt, wenn der konfigurierte Grenzwert zu groß ist, dann werden lange Impulse als kurze erkannt. Der Empfänger kann bei einer passenden Checksumme diese Fehler nicht erkennen. Im ersten Fall liefert der Empfänger möglicherweise Zeiten, die weit in der Zukunft liegen, und im zweiten Fall Zeiten aus der Vergangenheit.

VAR_INPUT

```
VAR_INPUT
    DCF_PULSE   :BOOL;
    RUN         :BOOL;
END_VAR
```

DCF_PULSE: Das DCF-77 Signal.
RUN: Eine steigende Flanke an diesem Eingang initialisiert den Funktionsbaustein und startet die Dekodierung des DCF-77 Signals. Wird dieser Eingang zurückgesetzt, dann wird die Dekodierung gestoppt

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        :BOOL;
    ERR         :BOOL;
    ERRID       :UDINT;
    ERRCNT      :UDINT;
    READY       :BOOL;
    CDT         :DATE_AND_TIME;
END_VAR
```

BUSY: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt.
ERR: Ist ein Fehler bei der Dekodierung aufgetreten, dann wird dieser Ausgang gesetzt.
ERRID: Liefert bei einem gesetzten ERR-Ausgang die Fehlernummer.
ERRCNT: Anzahl der aufgetretenen Fehler seit dem letzten fehlerfreien Empfang.
READY: Ist dieser Ausgang gesetzt, dann sind die Daten am CDT-Ausgang gültig.
CDT: Die DCF-77 Zeit im DATE_AND_TIME-Format.

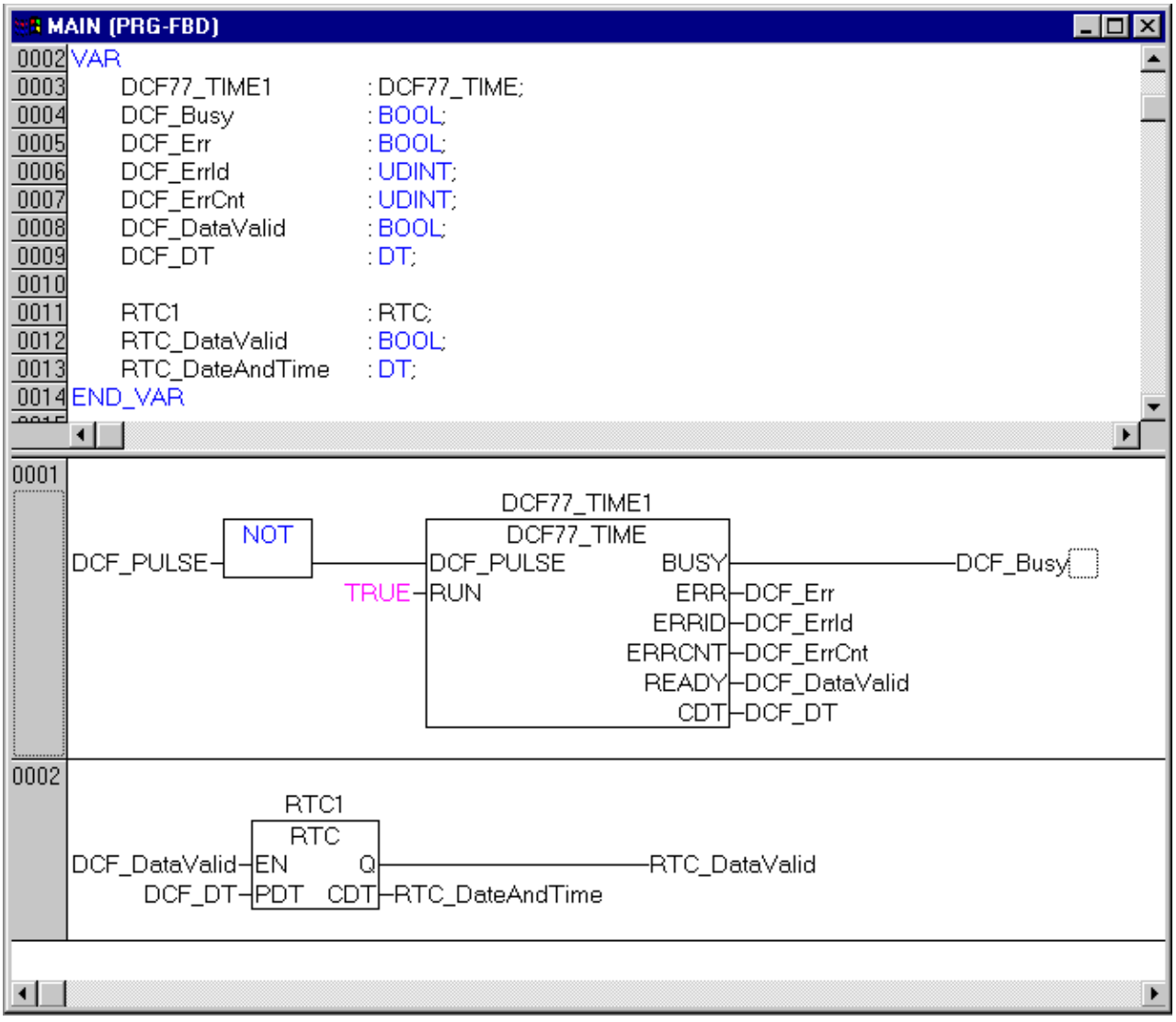
Fehlerbeschreibung:

Fehlercodes	Fehlerbeschreibung
0	kein Fehler
0x100	Timeout-Fehler. Möglicherweise kein DCF-77 Signal erkannt.
0x200	Parity-Fehler. In den empfangenen Daten wurden fehlerhafte Bits erkannt.

Fehlercodes	Fehlerbeschreibung
0x300	Fehlerhafte Daten empfangen. Da bei der Parity Prüfung nur ein falsches Bit erkannt werden kann, werden die empfangenen Daten noch auf Gültigkeit überprüft (bei Monat = 13 wird z.B. dieser Fehlerkode gesetzt).
0x400	Der letzte Dekodierungszyklus war zu lang. Dieser Fehler kann bei einem schlechten Empfang auftreten (zu wenig Sekundenmarken wurden empfangen).
0x500	Der letzte Dekodierungszyklus war zu kurz. Dieser Fehler kann bei einem schlechten Empfang auftreten (zusätzliche Flanken wurden empfangen).

Beispiel für einen Aufruf in FUP

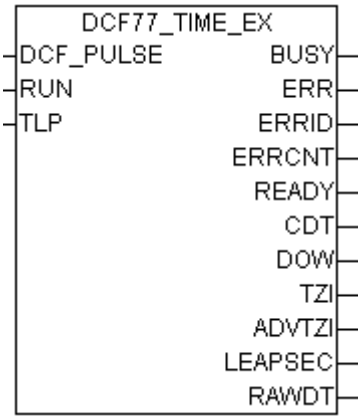
In der Beispielanwendung wird bei einem fehlerfreien Empfang die Real Time Clock mit der Funkzeit synchronisiert.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1313 und höher	PC or CX (x86, ARM)	TcUtilities.Lib

4.73 DCF77_TIME_EX



Mit dem Funktionsbaustein "DCF77_TIME_EX" kann das DCF-77 Funkuhr-Signal dekodiert werden. Im Gegensatz zu dem "DCF77_TIME [▶ 222]"-Funktionsbaustein werden bei diesem Baustein standardmäßig zwei aufeinanderfolgende Telegramme auf Plausibilität geprüft.

Über eine steigende Flanke am RUN-Eingang wird die Dekodierung gestartet und fortgesetzt, solange der RUN-Eingang gesetzt ist. Der Funktionsbaustein braucht im ungünstigsten Fall maximal eine Minute um sich zu synchronisieren, und zwei weitere Minuten um Daten für die nächste Minute zu dekodieren. Während dieser Zeit wird auf die fehlende 59. Sekundenmarke gewartet. Intern findet bei dem Funktionsbaustein eine Abtastung des DCF-77 Signals statt. Um die Flanken fehlerfrei abtasten zu können, sollte der Funktionsbaustein in jedem Zyklus der SPS einmal aufgerufen werden. Bei einer Zykluszeit <= 25ms können zufriedenstellende Ergebnisse erzielt werden. Bei einem fehlenden oder fehlerhaften DCF-77 Signal wird der ERR-Ausgang auf TRUE gesetzt und ein entsprechender Fehlercode am ERRID-Ausgang gesetzt. Bei dem nächsten fehlerfreien Empfang werden die ERR- und ERRID-Ausgänge zurückgesetzt. Manche der Empfänger liefern ein invertiertes DCF-77 Signal. In so einem Fall muss das Signal zuerst invertiert werden und dann an den DCF_PULSE-Eingang geleitet werden. Bei einem fehlerfreien Betrieb wird die aktuelle Zeit im Minutentakt am CDT-Ausgang aktualisiert. Dabei wird für einen Zyklus der SPS bei der Nullten Sekunde der READY-Ausgang auf TRUE gesetzt. In dieser Zeit ist die DCF-77 Zeit am CDT-Ausgang gültig und kann im SPS-Programm ausgewertet werden. Der READY-Ausgang wird nur dann gesetzt, wenn die Daten für die kommende Minute fehlerfrei empfangen werden konnten. Die Fehlererkennung wird mit Hilfe der übertragenen Paritätsbits durchgeführt und die zwei letzten Telegramme auf Plausibilität überprüft. Bei schlechten Empfangsverhältnissen kann eine zu 100% fehlerfreie Erkennung nicht gewährleistet werden. D.h. bei zwei fehlerhaften (invertierten) Bits in zwei danach folgenden Telegrammen kann der Funktionsbaustein keinen Fehler erkennen und setzt den READY-Ausgang ebenfalls auf TRUE. Durch die Plausibilitätsprüfung ist die Wahrscheinlichkeit, dass die passenden Bits verfälscht werden und ein solcher Fehler nicht erkannt wird recht gering.

Die auftretenden Fehler während des Empfangs werden von dem Funktionsbaustein registriert. Der ERRCNT-Ausgang ist ein Fehlerzähler. Der Zähler gibt Auskunft über die Anzahl der aufgetretenen Fehler seit dem letzten fehlerfreien Empfang. Bei dem nächsten fehlerfreien Empfang wird dieser Zähler zurückgesetzt.

Zeitcode

Während jeder Minute werden die Nummern von Jahr, Monat, Tag, Wochentag, Stunde, Minute BCD-codiert durch Impulsmodulation der Sekundenmarken übertragen. Die übertragenen Informationen gelten jeweils für die nachfolgende Minute. Jede Sekunde wird eine der Sekundenmarken übertragen. Dabei entspricht eine Sekundenmarke mit der Dauer von 0.1s einer binären Null und eine mit der Dauer von 0.2s einer binären Eins. Die Informationen werden mit 3 Prüfbits ergänzt. Bei der 59. Sekunde fehlt die Sekundenmarke und diese "Lücke" kann zur Synchronisation des Empfängers benutzt werden.

VAR_INPUT

```
VAR_INPUT
    DCF_PULSE   :BOOL;
    RUN         :BOOL;
    TLP         :TIME := 140ms;
END_VAR
```

DCF_PULSE: Das DCF-77 Signal.
RUN: Eine steigende Flanke an diesem Eingang initialisiert den Funktionsbaustein und startet die Dekodierung des DCF-77 Signals. Wird dieser Eingang zurückgesetzt, dann wird die Dekodierung gestoppt
TLP: Über diesen Eingang wird abhängig von der Spezifikation des Empfängers eine feste Grenze zur Unterscheidung zwischen Null und Eins festgelegt. Bei einem gestörten Signal sind die Impulsbreiten immer kleiner. In den Empfängerspezifikationen finden sich meistens Angaben über minimale und maximale Impulsbreiten der beiden logischen Signale, wobei breitere Impulse bei höheren Feldstärken zu erwarten sind und schmale Impulse bei niedrigen Feldstärken oder entsprechenden Störverhältnissen. In der Nähe des Senders (wo die Feldstärke sehr groß ist) können ebenfalls Probleme entstehen wenn die Impulsbreite der logischen Null zu groß wird. **Prüfen Sie deshalb die Spezifikation des eingesetzten Empfängers und konfigurieren entsprechend die Impulslänge.**

Zum Beispiel wird in der Spezifikation von Atmel T4227 (Time Code Receiver) folgende Pulslänge angegeben:
100ms Pulse (Null): Min: 70ms, Typical: 95ms, Max: **130ms**
200ms Pulse (Eins): Min **170ms**, Typical 195ms, Max 235ms
Für diesen IC wäre ein Grenzwert von 150ms optimal = 130 + ((170ms - 130ms) / 2).

Wenn der konfigurierte Grenzwert für die Impulslänge einen zu kleinen Wert hat, dann werden kurze Impulse als lange erkannt. Umgekehrt gilt, wenn der konfigurierte Grenzwert zu groß ist, dann werden lange Impulse als kurze erkannt. Der Empfänger kann bei einer passenden Checksumme diese Fehler nicht erkennen. Im ersten Fall liefert der Empfänger möglicherweise Zeiten, die weit in der Zukunft liegen, und im zweiten Fall Zeiten aus der Vergangenheit.

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: Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang gesetzt.
ERR: Ist ein Fehler bei der Dekodierung aufgetreten, dann wird dieser Ausgang gesetzt.
ERRID: Liefert bei einem gesetzten ERR-Ausgang die Fehlernummer.
ERRCNT: Anzahl der aufgetretenen Fehler seit dem letzten fehlerfreien Empfang.
READY: Ist dieser Ausgang gesetzt, dann sind die Daten am CDT-Ausgang gültig.
CDT: Die DCF-77 Zeit im DATE_AND_TIME-Format.
DOW: Wochentag nach ISO 8601: 1 = Montag... 7 = Sonntag.
TZI: Zeitzoneninformation (Sommer-/Winterzeit).
ADVTZI: Ankündigung eines MEZ -> MESZ oder MESZ -> MEZ Wechsels. Am Ende dieser Stunde wird MESZ/MEZ umgestellt (siehe Telegrammbeispiele).
LEAPSEC: Ankündigung einer Schaltsekunde. Am Ende dieser Stunde wird eine Schaltsekunde eingefügt (siehe Telegrammbeispiele).
RAWDT: Zuletzt decodierte (rohe) Bit-Information. Bitte beachten Sie, dass nur die Paritätsbits der Zeitinformation überprüft werden. Die Paritätsbits der Wetterdaten werden nicht ausgewertet!

Fehlerbeschreibung:

Fehlercodes	Fehlerbeschreibung
0	kein Fehler
0x100	Timeout-Fehler. Möglicherweise kein DCF-77 Signal erkannt.
0x200	Parity-Fehler. In den empfangenen Daten wurden fehlerhafte Bits erkannt.

Fehlercodes	Fehlerbeschreibung
0x300	Fehlerhafte Daten wurden empfangen. Da bei der Parity Prüfung nur ein falsches Bit erkannt werden kann, werden die empfangenen Daten noch auf Gültigkeit überprüft (bei Monat = 13 wird z.B. dieser Fehlerkode gesetzt).
0x400	Der letzte Dekodierungszyklus war zu lang. Dieser Fehler kann bei einem schlechten Empfang auftreten (zu wenig Sekundenmarken wurden empfangen).
0x500	Der letzte Dekodierungszyklus war zu kurz. Dieser Fehler kann bei einem schlechten Empfang auftreten (zusätzliche Flanken wurden empfangen).

Telegrammbeispiele:

MESZ -> MEZ (Sommerzeit -> Winterzeit)

'0 01110100100111 011001 00011011 0100001 011001 111 00001 000100000': Sonntag, 26.10.08 02:58:00, TZI = eTimeZoneID_Daylight, ADVTZI = TRUE
'0 11110001101110 011001 10011010 0100001 011001 111 00001 000100000': Sonntag, 26.10.08 02:59:00, TZI = eTimeZoneID_Daylight, ADVTZI = TRUE
'0 01000001001110 010101 00000000 0100001 011001 111 00001 000100000': Sonntag, 26.10.08 02:00:00, TZI = eTimeZoneID_Standard, ADVTZI = TRUE
'0 01111110100000 000101 10000001 0100001 011001 111 00001 000100000': Sonntag, 26.10.08 02:01:00, TZI = eTimeZoneID_Standard, ADVTZI = FALSE

MEZ -> MESZ (Winterzeit -> Sommerzeit)

'0 01000110111010 010101 00011011 1000001 000011 111 11000 000100000': Sonntag, 30.03.08 01:58:00, TZI = eTimeZoneID_Standard, ADVTZI = TRUE
'0 01000010100111 010101 10011010 1000001 000011 111 11000 000100000': Sonntag, 30.03.08 01:59:00, TZI = eTimeZoneID_Standard, ADVTZI = TRUE
'0 10000111100011 011001 00000000 1100000 000011 111 11000 000100000': Sonntag, 30.03.08 03:00:00, TZI = eTimeZoneID_Daylight, ADVTZI = TRUE
'0 01010000010110 001001 10000001 1100000 000011 111 11000 000100000': Sonntag, 30.03.08 03:01:00, TZI = eTimeZoneID_Daylight, ADVTZI = FALSE

Schaltsekunde

'0 10110000100001 000111 10011010 0000000 100000 001 10000 100100001': Donnerstag, 01.01.09 00:59:00, TZI = eTimeZoneID_Standard, LEAPSEC = TRUE
'0 11010010111000 000111 00000000 1000001 100000 001 10000 1001000010': Donnerstag, 01.01.09 01:00:00, TZI = eTimeZoneID_Standard, LEAPSEC = TRUE
'0 01000110011101 000101 10000001 1000001 100000 001 10000 100100001': Donnerstag, 01.01.09 01:01:00, TZI = eTimeZoneID_Standard, LEAPSEC = FALSE

: LEAPSEC bit;

: MEZ/MESZ-Information;

: ADVTZI bit;

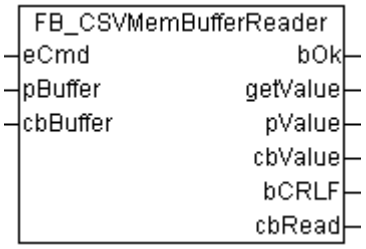
Beispiel:

Siehe in der Beschreibung des "DCF77_TIME [▶ 222]"-Funktionsbausteins.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1340 und höher TwinCAT v2.11.0 Build > 1542 und höher	PC or CX (x86, ARM)	TcUtilities.Lib

4.74 FB_CSVMemBufferReader



Mit diesem Funktionsbaustein können Datensätze die in einem externen Puffer liegen in einzelne Datenfelder zerlegt/interpretiert werden. Die Pufferdaten könnten z.B. zuerst mit Hilfe der Funktionsbausteine für den Dateizugriff aus einer Datei gelesen werden. Der Funktionsbaustein liest das erste oder das nächste Datenfeld und gibt dessen Wert entweder als String am *getValue*-Ausgang oder als Adresse/Bytegröße am *pValue*/*cbValue*-Ausgang zurück.

Die Daten im Puffer müssen einem bestimmten Format entsprechen damit sie von dem Funktionsbaustein richtig interpretiert werden können. Zur Trennung der Datensätze muss das Datensatztrennzeichen CRLF (CR=Carriage Return, LF=Line Feed) verwendet werden. Der letzte Datensatz muss ebenfalls mit einem CRLF abschließen. Einzelne Datenfelder müssen mit dem Datenfeldtrennzeichen getrennt worden sein. In der Standardeinstellung wird dafür das Datenfeldtrennzeichen Semikolon verwendet. Über die globale SPS-Variable **DEFAULT_CSV_FIELD_SEP** kann das Trennzeichen vom Semikolon auf Komma umkonfiguriert werden.

VAR_INPUT

```
VAR_INPUT
    eCmd        : E_EnumCmdType := eEnumCmd_First;
    pBuffer     : DWORD;
    cbBuffer    : UDINT;
END_VAR
```

eCmd: Steuerparameter [▶ 236] für den Pufferbaustein: eEnumCmd_First liest das erste Datenfeld, eEnumCmd_Next liest das nächste Datenfeld. Andere Parameterwerte werden nicht benutzt.

pBuffer: Adresse (Pointer) auf die Quellpuffervariable. Die Adresse kann mit dem ADR-Operator ermittelt werden. Dieser Puffer beinhaltet die zu lesenden Datensatz/Datenfeld-Daten.

cbBuffer: Die Bytegröße der zu interpretierenden Daten im Quellpuffer (Datensatz/Datenfeld-Daten). Die Puffergröße kann viel größer sein als die reine zu interpretierende Datenmenge. Bitte geben Sie hier nur die tatsächliche Länge der zu interpretierenden Daten an.

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 = Fehlerhafte Daten/fehlerhafte Eingangsparameter oder wenn das Ende der Daten erreicht wurde und kein weiteres Datenfeld gelesen werden konnte.

getValue: Das zuletzt gelesene Datenfeld als String. Bei Datenfeldern ohne Steuerzeichen und Binärdaten liefert dieser Ausgang das vollständige Datenfeld als Nullterminierter String. Datenfelder mit Steuerzeichen oder Binärdaten können aber an diesem Ausgang einen unvollständigen String zurückliefern. In diesem Fall werden die Ausgänge *pValue/cbValue* für den Zugriff auf das zuletzt gelesene Datenfeld benutzt.

pValue: Adresse (Pointer) auf das erste Datenbyte des Datenfelds. Bitte beachten Sie: Leere Datenfelder werden nicht wie bei einem SPS -String üblich mit einer Null terminiert und besitzen daher keine Daten. Die Adresse ist in diesem Fall Null.

cbValue: Datenfeldlänge in Bytes. Bitte beachten Sie: Leere Datenfelder werden nicht wie bei einem SPS-String üblich mit einer Null terminiert und besitzen daher keine Daten. Die Länge ist in diesem Fall auch Null.

bCRLF: Dieser Ausgang wird gesetzt wenn beim letzten Lesebefehl das Datensatzende erreicht wurde. Das zuletzt gelesene Datenfeld gehört noch zum vorherigen Datensatz. Das nächste Datenfeld gehört dann zu einem neuen Datensatz.

cbRead: Anzahl der erfolgreich gelesenen/interpretieren Datenbytes. Diese Zahl kann größer sein als die Datenfeldlänge am *cbValue*-Ausgang. Die Länge am *cbRead*-Ausgang beinhaltet auch die interpretierten Datenfeld-/Datensatztrennzeichen.

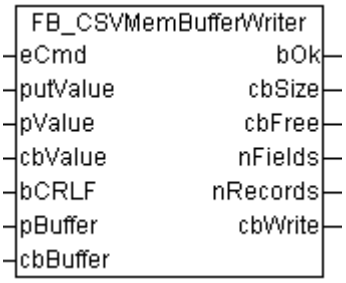
Beispiel:

Siehe: Beispiel: Schreiben/Lesen einer CSV-Datei [▶ 270].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

4.75 FB_CSVMemBufferWriter



Mit diesem Funktionsbaustein können Datensätze im CSV-Format aus einzelnen Datenfeldern in einem externen Puffer erzeugt werden. Der Inhalt des Puffers kann danach z.B. mit Hilfe der Bausteine für den Dateizugriff in eine Datei geschrieben werden. Das neue Datenfeld kann entweder über die *putValue*-Variable (String) oder über die optionalen *pValue*- und *cbValue*-Variablen an den Baustein übergeben werden. Dies hängt davon ab ob Sie Datenfelder ohne Steuerzeichen (String) oder Datenfelder mit Steuerzeichen oder Binärdaten in den Datensatz schreiben wollen. Der Funktionsbaustein kann mehrere Datensätze im Puffer erzeugen bis die maximal verfügbare Puffergröße erreicht ist. Das Ende eines Datensatzes (letztes Datenfeld im aktuellen Datensatz) wird an das Datenfeld automatisch angehängt wenn beim Schreiben des Datenfeldes die bCRLF-Variable auf TRUE gesetzt wurde. Die Datenfeldtrennzeichen werden vom Baustein automatisch hinzugefügt. In der Standardeinstellung wird dafür das Datenfeldtrennzeichen Semikolon verwendet. Über die globale SPS-Variable **DEFAULT_CSV_FIELD_SEP** kann das Trennzeichen vom Semikolon auf Komma umkonfiguriert werden.

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: Steuerparameter [▶ 236] für den Pufferbaustein: eEnumCmd_First fügt das erste Datenfeld in den Puffer, eEnumCmd_Next fügt das nächste Datenfeld hinzu. Andere Parameterwerte werden nicht benutzt.

putValue: Ein neues Datenfeld als String. Dieser Eingang muss ein Leerstring sein wenn statt dieses Eingangs die optionalen Parameter *pValue* und *cbValue* benutzt werden.

pValue: Optional: Adresse eines externen Bytepuffers in dem sich das neue Datenfeld befindet. Zusammen mit dem cbValue-Parameter kann dieser Eingang dann benutzt werden wenn z.B. ein Datenfeld mit Steuerzeichen oder Binärdaten in den Datensatz geschrieben werden soll. Die Steuerzeichen oder Binärdaten im Datenfeld könnten den *putValue*-String an einer ungewünschten Stelle abschneiden und werden deshalb als Bytepuffer übergeben. Dieser Eingang muss den Wert Null haben wenn er nicht benutzt wird.

cbValue: Optional: Länge der Datenfelddaten im externen Bytepuffer. Dieser Eingang muss den Wert Null haben wenn er nicht benutzt wird.

bCRLF: Wenn dieser Eingang gesetzt ist dann wird das neue Datenfeld mit einem CRLF Datensatztrennzeichen abgeschlossen. Datenfelder die danach geschrieben werden gehören dann zum neuen Datensatz.

pBuffer: Adresse (Pointer) auf die Zielpuffervariable. Die Adresse kann mit dem ADR-Operator ermittelt werden. In diesem Puffer erzeugt der Funktionsbaustein die Datensätze im CSV-Format.

cbBuffer: Maximal verfügbare Größe (in Byte) der Zielpuffervariablen. Die Größe kann mit dem SIZEOF-Operator ermittelt werden.

VAR_OUTPUT

```
VAR_OUTPUT
    bOk      : BOOL;
    cbSize   : UDINT;
    cbFree   : UDINT;
    nFields  : UDINT;
    nRecords : UDINT;
    cbWrite  : UDINT;
END_VAR
```

bOk: TRUE = Success, FALSE = Pufferüberlauf oder fehlerhafte Eingangsparameter.

cbSize: Der aktuelle Puffer-Füllstatus (Anzahl der im Puffer erzeugten Datenbytes).

cbFree: Anzahl der freien Datenbytes im Puffer.

nFields: Anzahl der geschriebenen Datenfelder.

nRecords: Anzahl der geschriebenen Datensätze.

cbWrite: Anzahl der zuletzt geschriebenen Datenbytes (die Länge des letzten Datenfelds + eventuelle Datensatz- oder Datenfeld-Trennzeichen).

Beispiel:

Siehe: Beispiel: Schreiben/Lesen einer CSV-Datei [▶ 270].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build > 1550	PC or CX (x86, ARM)	TcUtilities.Lib

5 Datentypen

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 : Das Jahr: 1970 ~ 2106;

wMonth : Der Monat: 1 ~ 12 (Januar = 1, Februar = 2 usw.);

wDayOfWeek : Der Wochentag: 0 ~ 6 (Sonntag = 0, Montag = 1 usw.);

wDay : Tag des Monats: 1 ~ 31;

wHour : Stunde: 0 ~ 23;

wMinute : Minute: 0 ~ 59;

wSecond : Sekunde: 0 ~ 59;

wMilliseconds : Millisekunde: 0 ~ 999;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 und höher	PC or CX (x86)	TcUtilities.Lib
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0 und höher	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.3 PROFILERSTRUCT

```
TYPE PROFILERSTRUCT:
STRUCT
   LastExecTime     :DWORD;
   MinExecTime      :DWORD;
   MaxExecTime      :DWORD;
   AverageExecTime  :DWORD;
   MeasureCycle     :DWORD;
END_STRUCT
END_TYPE
```

LastExecTime: Der letzte gemessene Wert der Ausführungszeit in [µs].

MinExecTime: Die minimale Ausführungszeit in [µs].

MaxExecTime: Die maximale Ausführungszeit in [µs].

AverageExecTime: Die mittlere Ausführungszeit der 10 letzten Messungen in [µs].

MeasureCycle: Anzahl der bereits durchgeführten Messungen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
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: Die Netzwerkadresse des Remote-PC's;

Name: Die Remote-PC-Bezeichnung;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.5 REMOTEPCINFOSTRUCT

```
TYPE REMOTEPCINFOSTRUCT
    ARRAY[0..99] OF REMOTEPC;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

Sehen Sie dazu auch

REMOTEPC [▶ 232]

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: Die tatsächliche Bytelänge des Symboleintrags in der Symboltabelle. Die Symbole werden in einer Symboltabelle abgelegt. Die Länge der einzelnen Einträge ist variabel und abhängig von der Länge des Symbolnamens, der Typbezeichnung und des Kommentars.

idxGroup: Der Index-Group der Symbolvariablen;

idxOffset: Der Index-Offset der Symbolvariablen;

byteSize: Die tatsächliche Speicherlänge, die der Wert der Symbolvariablen belegt in Byte. Eine boolsche SPS-Variable belegt z.B. einen Byte und ein String mit 20 Zeichen belegt tatsächlich 21 Byte (20 Byte für Zeichen + ein Byte für die abschließende Null);

adsDataType: Die ADS-Datentyp-Id [▶ 234]. Diese Typ-Bezeichnung wird beim ADS-Zugriff auf symbolische Variablen benutzt. Alle SPS-Strukturen und Arrays (selbstdefinierte Datentypen) besitzen die Ads-Datentypbezeichnung: ADST_ BIGTYPE und können über diese Datentypkonstante nicht identifiziert werden. Um die selbstdefinierten Datentypen identifizieren zu können, benutzen Sie die *symDataType*-Variable oder lesen Sie die Basistypen der einzelnen Variablen der Struktur.

symDataType: Die Datentypbezeichnung der Symbolvariablen als String.(T_MaxString)Z.B. der Typ-Name einer vom Benutzer definierten SPS-Datenstruktur (max. 255 Zeichen).

symComment: Der Kommentar zu der Symbolvariablen, den der Benutzer in der Zeile der SPS-Variablendefinition hinzugefügt hat (max. 255 Zeichen).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.7.0, Build > 519	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0, Build >723	PC or CX (x86)	TcUtilities.Lib
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, (* string of type T_MaxString! *)
    ARGTYPE_BOOL,
    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 >
1328 *)
    ARGTYPE_HUGE    (* signed 128 bit integer (T_HUGE_INTEGER), implemented in TwinCAT 2.10 Build >
1328 *)
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build >= 959	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.11 E_PersistentMode

Modus in dem die persistenten Daten geschrieben werden sollen. Siehe auch: Schreiben der pers. Daten: Systemverhalten [▶ 278].

```
TYPE E_PersistentMode :
(
    SPDM_2PASS      := 0,
    SPDM_VAR_BOOST  := 1
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0, Build >= 959	PC or CX (x86)	TcUtilities.Lib
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.13 E_EnumCmdType

Steuerparameter für Aufzählungsbausteine. Nicht alle Parameter werden von jedem Aufzählungsbaustein benutzt!

```
TYPE E_EnumCmdType :
(
    eEnumCmd_First := 0,
    eEnumCmd_Next,
    eEnumCmd_Abort
);
END_TYPE
```

eEnumCmd_First: Listet das erste Element auf.

eEnumCmd_Next: Listet das nächste Element auf.

eEnumCmd_Abort: Bricht die Auflistung ab (schließt geöffnete handles).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib
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
```

Die Transportschicht mit der AMS Nachrichten befördert werden. Zur Zeit wird nur TCP/IP als Transportschicht unterstützt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0, Build > 1033 TwinCAT v2.10.0, Build > 1257	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.15 E_NumGroupTypes

Numerische Zahlengruppen.

```
TYPE E_EnumCmdType :
(
    eNumGroup_Float,
    eNumGroup_Unsigned,
    eNumGroup_Signed
);
END_TYPE
```

eNumGroup_Float: Fließkommazahlen.

eNumGroup_Unsigned: Vorzeichenlose Zahlen.

eNumGroup_Signed: Vorzeichenbehaftete Zahlen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.16 E_MIB_IF_Type

Management Information Base Interface Typ.

```
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
```

Zusätzliche Informationen zur konfigurierten Zeitzone des Betriebsystems.

- eTimeZoneID_Invalid = die Zeitzone-Konfiguration konnte nicht gelesen werden;
- eTimeZoneID_Unknown = die Zeitzone-Konfiguration konnte zwar gelesen werden, die Normal-/ Sommerzeit-Information ist aber unbekannt.
- eTimeZoneID_Standard = es wird aktuell die Normalzeit benutzt;
- eTimeZoneID_Daylight = es wird aktuell die Sommerzeit benutzt;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1323	PC or CX (x86, ARM)	TcUtilities.Lib

5.19 E_DbgContext

Dieser Variablentyp kann von Protokollbausteinen verwendet werden und bestimmt den Kontext der Debugausgabe.

```
TYPE E_DbgContext:
(
    eDbgContext_NONE := 0,(* Not used *)
    eDbgContext_USER := 1,(* Service user *)
    eDbgContext_PROV := 2 (* Service provider *)
);
END_TYPE
```

Wert	Bedeutung
eDbgContext_NONE	Parameter wird nicht benutzt
eDbgContext_USER	Die Debugausgabe wurde ausgelöst durch den Service-User
eDbgContext_PROV	Die Debugausgabe wurde ausgelöst durch den Service-Provider

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build > 1537	PC or CX (x86, ARM)	TcUtilities.Lib

5.20 E_DbgDirection

Dieser Variablentyp kann von Pufferbausteinen oder Protokollbausteinen zur Konfiguration der Debugausgabe verwendet werden. Die Debugausgabe selbst kann z.B. mit Hilfe der ADSLOGSTR-Funktion realisiert werden.

In einem Ringpuffer könnte z.B. die Debugausgabe über die Variable auf folgende Weise gesteuert werden:

- Beim Wert *eDbgDirection_IN* oder *eDbgDirection_ALL* erfolgt die Debugausgabe wenn dem Puffer ein neuer Wert hinzugefügt wird;
- Beim Wert *eDbgDirection_OUT* oder *eDbgDirection_ALL* erfolgt die Debugausgabe wenn ein Wert aus dem Puffer entfernt wird;

```
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
```

Wert	Bedeutung
eDbgDirection_OFF	Deaktiviert die Debuagausgabe
eDbgDirection_IN	Aktiviert die Ausgabe der eingehenden Telegramme
eDbgDirection_OUT	Aktiviert die Ausgabe der ausgehenden Telegramme
eDbgDirection_ALL	Aktiviert die Ausgabe der eingehenden und ausgehenden Telegramme

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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 : Datentypkennung [▶ 234].

cbLen : Anzahl der Bytes, die im Speicher belegt werden.

pData : Adresspointer.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

5.22 T_FILETIME

```
TYPE T_FILETIME :
STRUCT
    dwLowDateTime   : DWORD;
    dwHighDateTime  : DWORD;
END_STRUCT
END_TYPE
```

Variablen von diesem Typ sind 64 bit Zahlen. Der Wert entspricht der Anzahl der 100-Nanosekunden-Intervalle seit dem 1 Januar 1601 (UTC).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

5.23 T_ULARGE_INTEGER

```
TYPE T_ULARGE_INTEGER :
STRUCT
    dwLowPart  : DWORD;
    dwHighPart : DWORD;
END_STRUCT
END_TYPE
```

Variablen von diesem Typ repräsentieren eine vorzeichenlose 64 bit Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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)	

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
```

Variablen von diesem Typ repräsentieren eine vorzeichenlose 128 bit Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1327	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_ULARGE_INTEGER [▶ 240]

5.25 T_LARGE_INTEGER

```
TYPE T_LARGE_INTEGER :
STRUCT
    dwLowPart  : DWORD;
    dwHighPart : DWORD;
END_STRUCT
END_TYPE
```

Variablen von diesem Typ repräsentieren eine vorzeichenbehaftete 64 bit Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
```

Variablen von diesem Typ repräsentieren eine vorzeichenbehaftete 128 bit Zahl.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1328	PC or CX (x86, ARM)	TcUtilities.Lib

Sehen Sie dazu auch

T_ULARGE_INTEGER [▶ 240]

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: Diese Membervariable beinhaltet den eigentlichen Wert der Festkommazahl (16 Bit Vor- und Nachkommastellen).

n: Anzahl der Nachkommastellen. Zulässiger Bereich: 0..15. Das höherwertigste Bit ist für das Vorzeichenbit reserviert.

status: Status-Flags (reserviert, zur Zeit nicht benutzt).

Variablen von diesem Typ repräsentieren eine 16 Bit vorzeichenbehaftete Festkommazahl. Dieser Datentyp wird oft bei Systemen benutzt die keine FPU-Einheit besitzen (z.B.: Mikrokontroller oder Geräte aus dem Bereich der Fernwirktechnik). Wenn z.B. über die serielle Schnittstelle Daten im Festkommazahlenformat ausgetauscht werden sollten, dann müssen diese Daten in das passende Format konvertiert werden.

Die Anzahl der Nachkommastellen wird passend zum benötigten Zahlenbereich und Auflösung gewählt. Bei 15 Nachkommastellen lassen sich z.B. Festkommazahlen im Bereich: -1..1-2^15 darstellen. Dies entspricht etwa dem Fließkommazahlenbereich: -1..0.999969482421875.

Im Gegensatz zu Fließkommazahlen ist die Auflösung der Festkommazahlen im gesamten Zahlenbereich konstant. Leider haben die Festkommazahlen einen kleineren darstellbaren Zahlenbereich. Vorsicht ist geboten bei mathematischen Operationen die einen positiven oder negativen Überlauf generieren können.

Beispiel 1:

Ein A/D-C liefert Messwerte als vorzeichenbehaftete 16 Bit Festkommazahlen mit 15 Nachkommastellen. Diese Messwerte wurden in die SPS eingelesen und sollen in LREAL-Datentyp konvertiert werden.

```
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 );
```

Beispiel 2:

Die Parameter eines Micro-Controllers sind vorzeichenbehaftete 16 Bit Festkommazahlen mit 8 Nachkommastellen. Die LREAL-Parameter in der SPS sollen in diesen Format konvertiert werden.

```
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 );
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1326	PC or CX (X86, ARM)	TcUtilities.Lib

5.28 T_HHASHTABLE

Ein Hash-Tabellen-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: Anzahl der belegten Elemente.

nFree: Anzahl der freien Elemente.

Das Hash-Tabellen-Handle wird von dem Funktionsbaustein: FB_HashTableCtrl [▶ 191] verwendet.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

5.29 T_HLINKEDLIST

Ein 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: Anzahl der belegten Elemente.

nFree: Anzahl der freien Elemente.

Das Linked-List-Handle wird von dem Funktionsbaustein: FB_LinkedListCtrl [▶ 192] verwendet.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build >= 1339 TwinCAT v2.11.0 Build >= 1524	PC or CX (x86, ARM)	TcUtilities.Lib

5.30 T_HashTableEntry

Ein Hash-Tabellen-Eintrag/Element.

```
TYPE T_HashTableEntry:
STRUCT
    key     : DWORD := 0;(* Entry key *)
    value   : DWORD := 0;(* Entry value *)
END_STRUCT
END_TYPE
```

key: Schlüssel (32 bit)

value: Wert (32 bit).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1332	PC or CX (x86, ARM)	TcUtilities.Lib

5.31 T_LinkedListEntry

Ein Linked-List-Knoten/Element.

```
TYPE T_LinkedListEntry:
STRUCT
    value   : DWORD := 0;(* Linked list value *)
END_STRUCT
END_TYPE
```

value: Wert (32 bit, auch Pointer sind möglich).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1031 TwinCAT v2.10.0 Build > 1235	PC or CX (x86)	TcUtilities.Lib
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: Symbolischer Name des entfernten TwinCAT Systems. Dieser Name kann frei vergeben werden. Die maximale Stringlänge ist durch eine Konstante (default: 31 Zeichen) begrenzt.

sNetID: Netzwerkadresse des entfernten TwinCAT Systems.

sAddress: Systemadresse bezogen auf die jeweilige Transportschicht. Bei TCP/IP als Transportschicht wird hier die IP-Adresse angegeben. Die maximale Stringlänge ist durch eine Konstante (default: 79 Zeichen) begrenzt.

eTransport: Die Transportschicht [▶ 236] mit der AMS Nachrichten befördert werden. Zur Zeit wird nur die Transportschicht TCP/IP unterstützt.

tTimeout: Timeoutzeit. (zur Zeit reserviert und nicht benutzt).

dwFlags: Zusätzliche Optionen (zur Zeit reserviert und nicht benutzt).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.9.0 Build > 1033 TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcUtilities.Lib
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: Nullterminierter String mit dem Namen der Datei oder des Verzeichnisses.

sAlternateFileName: Nullterminierter String mit dem Alternativnamen der Datei oder des Verzeichnisses im klassischen 8.3 Format (filename.ext).

fileAttributes: Struktur [▶ 245] mit Datei-/Verzeichnis-Attributen.

fileSize: Bytegröße der Datei (64 bit Zahl [▶ 240]).

creationTime: Die Strukturvariable [▶ 240] gibt an wann die Datei oder das Verzeichnis erstellt wurde.

lastAccessTime: Bei einer Datei gibt die Struktur an wann die Datei zuletzt gelesen oder geschrieben wurde. Beim Verzeichnis gibt die Struktur an wann das Verzeichnis erstellt wurde.

lastWriteTime: Bei einer Datei gibt die Struktur an wann in die Datei zuletzt geschrieben wurde. Beim Verzeichnis gibt die Struktur an wann das Verzeichnis erstellt wurde.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1302	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.35 ST_FileAttributes

Datei- oder Verzeichnis-Attribute.

```
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: Die Datei oder Verzeichnis hat nur einen Lesezugriff. Die Datei kann von Applikationen gelesen werden, sie kann aber nicht beschrieben oder gelöscht werden. Im Falle eines Verzeichnisses können die Applikationen das Verzeichnis nicht löschen.

bHidden: Die Datei oder Verzeichnis ist versteckt und wird nicht angezeigt in einer Standardauflistung.

bSystem: Die Datei oder Verzeichnis gehört zum Teil des Betriebssystems oder wird exklusiv vom Betriebssystem benutzt.

bDirectory: Über dieses Attribut wird ein Verzeichnis identifiziert.

bArchive: Die Datei oder Verzeichnis gehören zum Archiv. Applikationen benutzen dieses Attribut um die Dateien für ein Backup oder für das Entfernen zu markieren.

bDevice: Reserviert.

bNormal: Die Datei oder Verzeichnis hat keine anderen Attribute gesetzt. Dieses Attribut ist nur dann gültig wenn es allein benutzt wird.

bTemporary: Die Datei wird nur kurzzeitig für die Aufbewahrung der Daten benutzt.

bSparseFile: Die Datei ist eine abgespeckte Datei.

bReparsePoint: Mit der Datei oder Verzeichnis wurde ein "reparse point" assoziiert.

bCompressed: Die Datei oder Verzeichnis ist komprimiert. Bei der Datei sind die Daten komprimiert und beim Verzeichnis ist die Komprimierung per Default aktiv für neu erstelle Dateien oder Unterverzeichnisse.

bOffline: Die Datei ist nicht immer verfügbar.

bNotContentIndexed: Die Datei ist nicht indiziert beim Indizierservice.

bEncrypted: Die Datei oder das Verzeichnis ist verschlüsselt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1302	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.36 ST_IPAdapterInfo

Netzwerkadapterinformationen.

```
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: Dieser Variable wird zur Zeit **nur unter Windows CE** verwendet! Beim TRUE wird der Netzwerkadapter als Default-Adapter von TwinCAT verwendet.

sAdapterName: Adaptername als String.

sDescription: Adapterbeschreibung als String.

physAddr: Physikalische Hardwareadresse [▶ 247].

dwIndex: Interner Adapter-Systemindex.

eType: Adapter-Typ [▶ 237].

sIpAddr: IP-Adresse. (T_IPv4Addr)

sSubNet: IP-Netzmaske.

sDefGateway: IP-Adresse des Default-Gateways.

bDhcpEnabled: Gibt an ob DHCP für diesen Adapter aktiviert wurde oder nicht.

sDhcpSrv: IP-Adresse des DHCP-Servers.

bHaveWins: Gibt an ob Windows Internet Name Service (WINS) verwendet wird oder nicht.

sPrimWinsSrv: IP-Adresse des primären WINS-Servers.

sSecWinsSrv: IP-Adresse des sekundären WINS-Servers.

tLeaseObt: Gibt an wann die IP-Adresse vom DHCP-Server „gemietet" wurde (UTC).

tLeaseExp: Gibt an wie lange die IP-Adresse vom DHCP-Server „vermietet“ werden darf bevor eine „Verlängerung" vom DHCP-Server beantragt werden muss (UTC).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.37 ST_IPAdapterHwAddr

Physikalische Adresse (MAC).

```
TYPE ST_IPAdapterHwAddr
STRUCT
    length  : UDINT := 0;
    b       : ARRAY[0..MAX_ADAPTER_ADDRESS_LENGTH] OF BYTE;
END_STRUCT
END_TYPE
```

length: Bytelänge der physikalischen Hardwareadresse.

b: MAC-Adressbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcUtilities.Lib

5.38 ST_FileRBufferHead

Ringpufferdatei-Header-Status. Diese Struktur wird von dem Funktionsbaustein FB_FileRingBuffer [▶ 166] verwendet. Beim Offnen der Ringpufferdatei wird diese Struktur gelesen und beim Schließen in die Ringpufferdatei gespeichert. Beim Lesen/Schreiben der Datensätze wird diese Struktur immer aktualisiert.

```
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 => Datei ist geöffnet, Bit 0 = 0 => Datei ist geschlossen. Bit 1 = 1 => Datei ist korrupt (wurde vorher nicht richtig geschlossen oder die maximale Puffergröße passt nicht).

acces: Zugriffszähler. Beim jedem erneuten Öffnen der Datei wird dieser Zähler inkrementiert.

nID: Benutzerdefinierter 32 Bit Wert.

cbBuffer: Die max. Ringpuffer-Dateigröße.

nCount: Aktuelle Anzahl der gespeicherten Datensätze.

cbSize: Aktuelle Anzahl der gespeicherten Datenbytes.

ptrFirst: Dateizeiger-Position des ältesten Datensatzes.

ptrLast: Dateizeiger-Position des neuesten Datensatzes.

rsrv0..rsrv3: Reserviert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.10.0 Build > 1313	PC oder CX (x86, ARM)	TcUtilities.Lib

5.39 ST_TimeZoneInformation

Zeitzone-Informationen.

```
TYPE ST_TimeZoneInformation :
STRUCT
    bias : DINT; (* Specifies the current bias, in minutes, for local time translation on computer.
        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
```

Die Normalzeit wird auch als Winterzeit bezeichnet. Die bias-Parameter können auch negative Werte annehmen.

bias: Definiert die aktuelle Differenz der Lokalzeit zur UTC-Zeit in Minuten. UTC = local time + bias.

standardName: Bezeichnung der Normalzeit als String.

standardDate: Diese Struktur [▶ 231] beinhaltet Informationen zum Übergang von Sommerzeit zur Normalzeit. Der Strukturparameter *wMonth* ist Null wenn dieser Wert nicht benutzt wird. Wenn dieser Parameter benutzt wird, dann muss auch der *daylightDate*-Parameter benutzt werden. Um *standardDate* konfigurieren zu können setzen Sie den *wYear*-Parameter gleich Null, bei *wDayOfWeek* wählen Sie den gewünschten Wochentag und bei *wDay* einen Wert zwischen 1 und 5 (Woche im Monat, 5 entspricht der letzten Woche).

standardBias: Zeitdifferenz in Minuten für Berechnungen der Lokalzeit während der Normalzeit. Dieser Wert ist meistens Null.

daylightName: Bezeichnung der Sommerzeit als String.

daylightDate: Diese Struktur beinhaltet Informationen zum Übergang von Normalzeit zur Sommerzeit. Der Strukturparameter *wMonth* ist Null wenn dieser Wert nicht benutzt wird. Wenn dieser Parameter benutzt wird, dann muss auch der *standardDate*-Parameter benutzt werden. Um *daylightDate* konfigurieren zu können setzen Sie den *wYear*-Parameter gleich Null, bei *wDayOfWeek* wählen Sie den gewünschten Wochentag und bei *wDay* einen Wert zwischen 1 und 5 (Woche im Monat, 5 entspricht der letzten Woche).

daylightBias: Zeitdifferenz in Minuten für Berechnungen der Lokalzeit während der Sommerzeit.

Beispiel:

FB_SetTimeZoneInformation [▶ 179].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
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
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >=2248	PC or CX (x86, ARM)	TcUtilities.Lib

6 Globale Variablen/Konstanten

6.1 Globale Variablen

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 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	TcPlcUtilities.Lib; Standard.Lib; PLCSystem.Lib; TcPLCAds.Lib; PLCHelper.Lib;
TwinCAT v2.8.0	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

6.2 Format Fehlercodes

Folgende Fehlercodes werden von dem Funktionsbaustein: FB_FormatString [▶ 156] oder der Funktion: F_FormatArgToStr [▶ 54] zurückgeliefert. Bei mehreren Argumenten wird zusätzlich zu dem Fehlercode die Argumentnummer (1..9) zurückgeliefert. Die Argumentnummer gibt Auskunft über die Stelle an der ein Fehler während der Formatierung festgestellt wurde.

Error code	Bedeutung
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)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

6.3 Fehler-Codes TcUtilities

Die folgenden Fehler-Codes werden von Funktionsbaustein FB_ScopeServerControl zurückgegeben.

```
TYPE E_UTILITIES_ERRORCODES :
(
   eUtilError_NoError := 0,
   eUtilError_ScopeServerNotAvailable := 16#8001, (*scope server not installed *)
   eUtilError_ScopeServerStateChange := 16#8002
);
END_TYPE
```

Fehler-Code	Enum	Bedeutung
0x0000	eUtilError_NoError	kein Fehler
0x8001	eUtilError_ScopeServerNotAvailable	TwinCAT Scope Server ist nicht verfügbar. Vielleicht ist er nicht installiert
0x8002	eUtilError_ScopeServerStateChange	Der angeforderte Statuswechsel ist nicht zulässig. Schlagen Sie die zulässigen Statuswechsel in B_ScopeServerControl state diagram nach.

7 Beispiele

7.1 Beispiel: Kommunikation BC/BX<->PC/CX (F_SwapRealEx)

Hier können Sie die kompletten Sourcen PC oder CX (x86) und BC (165) entpacken: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851042059.zip und https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851043467.zip

Das Beispiel zeigt die Verwendung der F_SwapRealEx-Funktion. Laden Sie die PC (i386, pro-Datei) Applikation in das Laufzeitsystem eines Programmier-PCs (i386) und die BC (165) Applikation (165, pr6-Datei) in das Laufzeitsystem eines Bus-Controllers (z.B. BC9000).

Die PC Applikation liest/schreibt zyklisch eine Strukturvariable aus/in den Merkerbereich des BCs. Die Strukturvariable beinhaltet REAL-Elemente. Diese müssen vor der Verwendung auf dem PC oder vor dem Versand zum BC in das richtige Format konvertiert werden. Damit auf den BC über ADS zugegriffen werden kann muss dieser als Remote PC in die Liste der TwinCAT AMS Router-Verbindungen eingetragen werden (rechter Mausklick auf das TwinCAT-System Icon -> Eigenschaften -> AMS Router).

Der BC und PC hat ein unterschiedliches Datenalignment. Definieren Sie Strukturen mit 4- oder besser mit 8-Byte-Alignment, wenn Sie diese für den Datenaustausch BC <-> PC verwenden wollen.

Die Strukturvariablendefinition auf beiden Systemen:

```
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 oder CX (x86) Applikation:

Vor dem Download muss die AMS-NetID des BCs im Programmcode konfiguriert werden. Nach erfolgreichem Lesevorgang wird zuerst die Länge der Daten dann die eine einfache Checksumme überprüft. Danach werden die REAL-Elemente in das PC-Format konvertiert. Steigende Flanke an der bWrite-Variablen aktiviert den Lesebefehl und steigende Flanke an der bRead-Variablen aktiviert den Schreibbefehl.

```
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) Applikation:

Nach jedem Schreibzugriff vom PC wird die Checksumme überprüft. Danach werden neue Zufallswerte generiert die zusätzlich mit einer einfachen Checksumme versehen werden.

```
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 Beispiel: Dateisuche (FB_EnumFindFileEntry, FB_EnumFindFileList)

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851044875.zip

Beispiel für FB_EnumFindFileEntry:

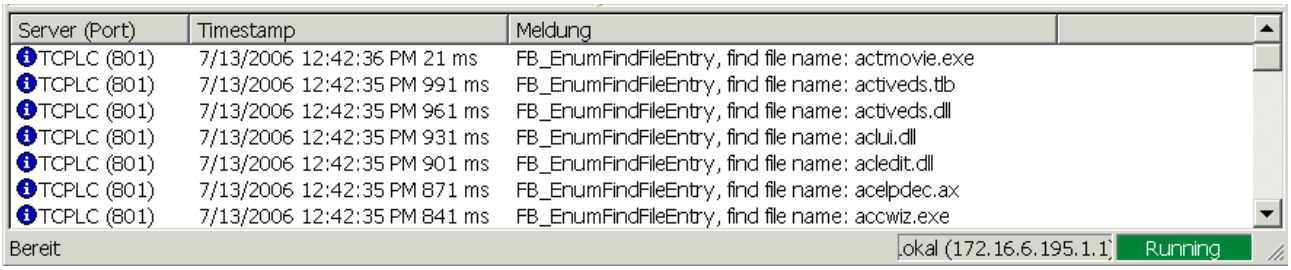
Auf dem lokalen TwinCAT System sollen alle Dateien im Verzeichnis C:\Windows\System32\ aufgelistet werden. Die Dateinamen sollen als Meldungen ins TwinCAT System Manager Logger Ausgabe geschrieben werden. Es soll möglich sein diesen Vorgang abzubrechen.

```
PROGRAM P_TestEnumEntry
VAR
    fbEnum : FB_EnumFindFileEntry := ( sNetID := '', tTimeout := T#5s, sPathName := 'C:
\Windows\System32\*.*' );
    bEnum : BOOL;
    bAbort : BOOL;
    nState : BYTE;
END_VAR
```

Bei einer steigenden Flanke an der *bEnum*-Variablen beginnt die Auflistung der gefundenen Dateien. Bei einer steigenden Flanke an der *bAbort*-Variablen wird der Vorgang abgebrochen.

```
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
```

Die geschriebenen Logmeldungen in der TwinCAT System Manager Logger Ausgabe:



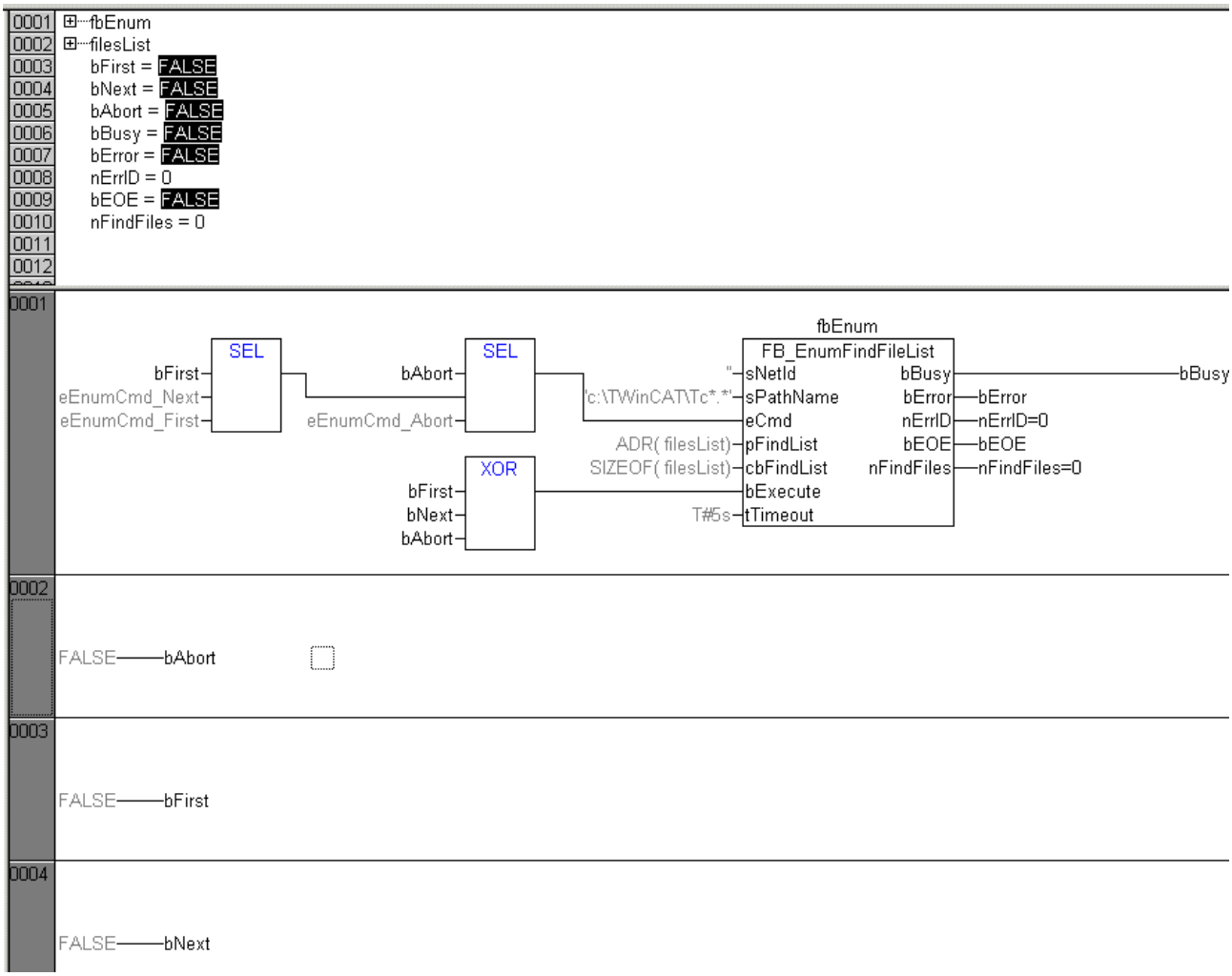
Server (Port)	Timestamp	Meldung
TCPLC (801)	7/13/2006 12:42:36 PM 21 ms	FB_EnumFindFileEntry, find file name: actmovie.exe
TCPLC (801)	7/13/2006 12:42:35 PM 991 ms	FB_EnumFindFileEntry, find file name: activeds.tlb
TCPLC (801)	7/13/2006 12:42:35 PM 961 ms	FB_EnumFindFileEntry, find file name: activeds.dll
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 | lokal (172.16.6.195.1.1) | Running

Beispiel für 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
```

Onlineansicht:



7.3 Beispiel: Datei-Ring-Fifo (FB_FileRingBuffer)

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851046283.zip

Folgendes Beispiel zeigt eine einfache Verwendung des Funktionsbausteins. Die steigende Flanke am *bOpen* öffnet eine existierende Ringpufferdatei. Wenn die Datei nicht existiert, wird eine neue erzeugt. Die steigende Flanke am *bClose* schließt eine geöffnete Datei. Die steigende Flanke am *bCreate* erzeugt eine neue Datei. Wenn Sie *bAdd* = TRUE setzen wird ein neuer Datensatz in die Ringpufferdatei geschrieben und beim *bRemove*=TRUE wird der älteste Datensatz entfernt.

```
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 Beispiel: Memory-Ring-Fifo (FB_MemRingBuffer)

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851047691.zip

Folgendes Beispiel zeigt eine einfache Verwendung des Funktionsbausteins. Es sollen Datensätze mit der gleichen Länge gepuffert werden (dies ist aber nicht zwingend notwendig). Die Datensätze haben folgende Struktur:

```
TYPE ST_DataSetEntry :
STRUCT
    bFlag : BOOL;
    nValue : BYTE;
    sMsg : STRING(20) := 'Unknown';
END_STRUCT
END_TYPE
```

Die Schnittstelle des FB_DataSetFifo-Funktionsbausteins:

Der im Beispielprojekt verwendete applikationsspezifische Funktionsbaustein FB_DataSetFifo benutzt intern den FB_MemRingBuffer-Funktionsbaustein. Dieser Baustein vereinfacht das Hinzufügen/Entfernen der Datensätze. Außerdem liefert der neue Funktionsbaustein den aktuellen prozentualen Füllstatus des Puffers und eine Overwrite-Option. Wenn *bOverwrite*-Eingang gesetzt ist und der Puffer bereits voll ist, dann wird der älteste Eintrag aus dem Puffer entfernt und mit dem neuen überschrieben.

```
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
```

Das Hauptprogramm:

Die steigende Flanke am *bReset* löscht alle Puffer Einträge. Wenn Sie *bAdd* = TRUE setzen wird ein neuer Datensatz in den Ringpuffer geschrieben und beim *bRemove*=TRUE wird der älteste Datensatz entfernt. Bei einer steigenden Flanke am *bGet* wird der älteste Datensatz gelesen aber nicht entfernt.

```
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 Beispiel: Memory-Ring-Fifo (FB_MemRingBufferEx)

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851049099.zip

Bei einer steigenden Flanke am bAdd werden neue Datenelemente (pubObj-Array) in den Ringpuffer abgelegt. Über eine steigende Flanke am bGet kann das älteste Datenelement in die getObj-Variable kopiert werden. Die nicht benötigten Datenelemente werden über eine steigende Flanke am bRelease aus dem Puffer entfernt.

```
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 Beispiel: Hash-Tabelle (FB_HashTableCtrl)

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851050507.zip

Das Beispielprojekt besitzt zwei Programmteile:

- P_TABLE_OF_UDINT ist eine einfaches Beispielprogramm welches nur 32 bit Werte in der Hash-Tabelle bearbeitet.
- Im P_TABLE_OF_STRUCTDATA wird gezeigt wie andere Datentypen (z.B. strukturierte Datentypen) in der Hash-Tabelle bearbeitet werden können.

Die maximale Anzahl der Tabellenelemente kann zur Laufzeit nicht verändert werden und wird in dem Beispielprojekt durch die MAX_DATA_ELEMENTS begrenzt. Wenn Sie mehr Elemente benötigen, dann müssen Sie das table-Array entsprechend vergrößern (d.h. den Wert der Konstante erhöhen).

```
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

Im ersten SPS-Zyklus werden in der Tabelle als Beispiel Informationen über Artikelnummer und Artikelname abgelegt. Die Artikelnummer dient als Schlüssel und der Array-Index des Artikelnamens als Wert.
Über eine steigende Flanke am bLookup kann der Artikelname anhand der Artikelnummer gefunden werden.

```
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

Dieses Programmteil zeigt wie strukturierte Datensätze statt einfacher 32-Zahlen in der Tabelle bearbeitet werden können. Dabei wird der 32 bit Elementwert nur als Referenzpointer auf den tatsächlichen Elementwert verwendet. Der Referenzpointer kann dann auch auf Instanzen von strukturierten Variablen oder anderen Datentypen zeigen. Die Funktionalität wurde in einem Funktionsblock gekapselt. Dieser Funktionsblock *FB_SpecialHashTableCtrl* kann als eine spezialisierte Version des *FB_HashTableCtrl*-Funktionsbausteins bezeichnet werden. Der *FB_HashTableCtrl*-Baustein wird auch intern von dem spezialisierten FB verwendet.

Die Funktion *DATAELEMENT_TO_STRING* wird nur verwendet um eine visuelle Ausgabe der Knotenwerte zu ermöglichen.

Als Beispiel wird die Strukturierte Variable vom Typ: *ST_DataElement* verwendet. Der Clou: Sie können die Datentypdeklaration von ST_DataElement um weitere Membervariablen erweitern, ohne dass Sie an dem Programm oder dem FB_SpecialHashTableCtrl-Funktionsbaustein Veränderungen durchführen müssen.

Die Typdeklaration von 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
```

Wie werden die 32 bit Elementwerte zu Referenzpointern auf die ST_DataElement-Array-Instanzen?

Die max. Größe der Tabelle ist durch MAX_DATA_ELEMENTS-Konstante begrenzt. Folglich können in der Tabelle nur MAX_DATA_ELEMENTS-Referenzpointer gespeichert werden. Der *FB_SpecialHashTableCtrl*-Baustein besitzt intern eine ST_DataElement-Array-Variable mit derselben Array-Größe wie die T_HashTableEntry-Array-Varriable. Zur Vereinfachung sind die Array-Indizies in beiden Arrays gleich!

Jedes T_HashTableEntry-Array-Element kann nur einmal in die Tabelle "reingehängt" werden. Dabei sucht der *FB_HashTableCtrl*-Funktionsbaustein nach einem freien/unbenutzten T_HashTableEntry-Array-Element und fügt es bei Erfolg der Tabelle zu. Mit Hilfe der Aktion *A_GetIndexAtPosPtr* kann der verwendete Index des T_HashTableEntry-Arrays ermittelt werden. Im nächsten Schritt wird dem zuletzt hinzugefügten 32 bit Knotenwert die Adresse desselben Arrayelements vom ST_DataElement-Array zugewiesen. Im Beispielprojekt durch den zweiten Aktionsaufruf: *A_Add*.

nodes[index].value := ADR(dataPool[index])

Die Zuweisung wird z.B. in der *FB_SpecialHashTableCtrl->A_Add*-Aktion durchgeführt:

```
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 Beispiel: Verkettete-Liste (FB_LinkedListCtrl)

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851051915.zip

Das Beispielprojekt besitzt zwei Programmteile:

- P_LIST_OF_UDINT ist eine einfaches Beispielprogramm welches nur 32 bit Werte in der verketteten Liste bearbeitet.
- Im P_LIST_OF_STRUCTDATA wird gezeigt wie andere Datentypen (z.B. strukturierte Datentypen) in der verketteten Liste bearbeitet werden können.

Die maximale Anzahl der Knotenelemente kann zur Laufzeit nicht verändert werden und wird in dem Beispielprojekt durch die MAX_DATA_ELEMENTS begrenzt. Wenn Sie mehr Knoten benötigen, dann müssen Sie das nodes-Array entsprechend vergrößern (d.h. den Wert der Konstante erhöhen).

```
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

Im ersten SPS-Zyklus wird das Handle der verketteten Liste initialisiert. Dieses Handle wird dann bei den Zugriffen auf die Liste als VAR_IN_OUT-Variable an den FB_LinkedListCtrl-Funktionsbaustein übergeben. Die verkettete Liste wird über die Aktionsaufrufe des FBs manipuliert. Auf diese Weise können Knotenelemente hinzugefügt/entfernt/durchsucht werden. Bei einer steigenden Flanke an der entsprechenden boolschen Variable wird die gewünschte Aktion ausgeführt. Wenn Sie das Programm starten werden alle Operationen einmalig ausgeführt.

```
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

Dieses Programmteil zeigt wie strukturierte Datensätze statt einfacher 32-Zahlen in der Liste bearbeitet werden können. Dabei wird der 32 bit Knotenwert nur als Referenzpointer auf den tatsächlichen Knotenwert verwendet. Der Referenzpointer kann dann auch auf Instanzen von strukturierten Variablen oder anderen Datentypen zeigen. Die Funktionalität wurde in einem Funktionsblock gekapselt. Dieser Funktionsblock *FB_SpecialLinkedListCtrl* kann als eine spezialisierte Version des *FB_LinkedListCtrl*-Funktionsbausteins bezeichnet werden. Der *FB_LinkedListCtrl*-Baustein wird auch intern von dem spezialisierten FB verwendet.

Die Funktion *DATAELEMENT_TO_STRING* wird nur verwendet um eine visuelle Ausgabe der Knotenwerte zu ermöglichen.

Als Beispiel wird die Strukturierte Variable vom Typ: *ST_DataElement* verwendet. Der Clou: Sie können die Datentypdeklaration von ST_DataElement um weitere Membervariablen erweitern, ohne dass Sie an dem Programm oder dem FB_SpecialLinkedListCtrl-Funktionsbaustein Veränderungen durchführen müssen.

Die Typdeklaration von 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
```

Es wurde eine einfache Suchfunktion implementiert. Sie können nach Knoten mit einem bestimmten *name*, *number* oder *price* suchen.

Wie werden die 32 bit Knotenwerte zu Referenzpointern auf die ST_DataElement-Array-Instanzen?

Die max. Größe der Liste ist durch MAX_DATA_ELEMENTS-Konstante begrenzt. Folglich können in der Liste nur MAX_DATA_ELEMENTS-Referenzpointer gespeichert werden. Der *FB_SpecialLinkedListCtrl*-Baustein besitzt intern eine ST_DataElement-Array-Variable mit derselben Größe wie die T_LinkedListEntry-Array-Varriable. Zur Vereinfachung sind die Array-Indizies in beiden Arrays gleich!

Jedes T_LinkedListEntry-Array-Element kann nur einmal in die Liste "reingehängt" werden. Dabei sucht der *FB_LinkedListCtrl*-Funktionsbaustein nach einem freien/unbenutzten T_LinkedListEntry-Array-Element und fügt es bei Erfolg der Liste zu. Mit Hilfe der Aktion *A_GetIndexAtPosPtr* kann der verwendete Index des T_LinkedListEntry-Arrays ermittelt werden. Im nächsten Schritt wird dem zuletzt hinzugefügten 32 bit Knotenwert die Adresse desselben Arrayelements vom ST_DataElement-Array zugewiesen. Im Beispielprojekt durch den Aktionsaufruf: *A_SetValueAtPosPtr*.

nodes[index].value := ADR(dataPool[index])

Die Zuweisung wird z.B. in der *FB_SpecialLinkedListCtrl->A_AddHeadValue*-Aktion durchgeführt:

```
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 Beispiel: Schreiben/lesen einer CSV-Datei

Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851053323.zip

CSV steht für Comma-Separated Values. Folgende Dokumentation beschreibt wie CSV-Dateien mit Hilfe der SPS-CSV-Hilfsfunktionen geschrieben bzw. gelesen werden können. In den CSV-Dateien, die eigentlich Textdateien sind, können einfach strukturierte Datensätze gespeichert und zum Datenaustausch zwischen zwei Systemen verwendet werden. Dieses Format erlaubt eine Speicherung von unterschiedlich langen Tabellen oder Listen. Eine Zeile in der Tabelle entspricht einem Datensatz (auch Zeile) in der CSV-Datei. Eine Zelle in einer Tabelle entspricht einem Datenfeld in der CSV-Datei.

Allgemeine Informationen zum unterstützten CSV-Format

- Die Dateien im CSV-Format sollten die Endung **.csv** besitzen.
- Das CRLF-Zeichen (CR =Carriage Return, LF=Line Feed) wird zur Trennung der einzelnen Datensätze (Zeilen) verwendet (Windows-Betriebssysteme). D.h. nach jedem Datensatz muss ein CRLF folgen.
- Das Ende der CSV-Datei muss ebenfalls mit CRLF-Zeichen abgeschlossen werden.
- Binärdaten müssen in Hochkommas eingeschlossen werden. Wenn keine Hochkommas verwendet werden dann dürfen im Datenfeld nur Zahlen und/oder Buchstaben verwendet werden.
- Felder mit Sonderzeichen/Steuerzeichen im Datenfeld werden in Anführungszeichen eingeschlossen. Wenn ein Einführungszeichen selbst im Datenfeld enthalten ist wird dieses verdoppelt.
- Ein spezielles Zeichen wird zur Trennung von Datenfeldern (Spalten) verwendet. Als Trennzeichen für die einzelnen Datenfelder wird von den Hilfsfunktionen standardmäßig ein Semikolon verwendet. In Deutschland und Europa wird als Datenfeldtrennzeichen ein Semikolon verwendet, in USA eher ein Komma. Über die globale SPS-Variable: **DEFAULT_CSV_FIELD_SEP** kann das Trennzeichen vom Semikolon auf Komma umkonfiguriert werden.
- Jeder Datensatz sollte die gleiche Anzahl an Datenfeldern (Spalten) besitzen.

Prinzipieller Aufbau einer CSV-Datei mit n-Spalten und n-Zeilen (die CRLF-Zeichen sind normalerweise nicht sichtbar und in der Abbildung vereinfacht mit den Buchstaben: **CRLF** dargestellt)

```
"Field1Record1";"Field2Record1";     ...     ;"Field(n)Record1"CRLF
"Field1Record2";"Field2Record2";    ...     ;"Field(n)Record2"CRLF

...

"Field1Record(n)";"Field2Record(n)";    ...     ;"Field(n)Record(n)"CRLF
```

Verfügbare Funktionsbausteine und Funktionen

- STRING_TO_CSVFIELD [▶ 70], ARG_TO_CSVFIELD [▶ 71]: Konvertiert SPS-Daten in ein Datenfeld im CSV-Format;
- CSVFIELD_TO_STRING [▶ 67], CSVFIELD_TO_ARG [▶ 69]: Konvertiert Datenfeld im CSV-Format in SPS-Daten;

- FB_CSVMemBufferWriter [▶ 229]: Generiert aus mehreren Datenfeldern Datensätze in einem Bytepuffer;
- FB_CSVMemBufferReader [▶ 228]: Zerlegt Datensätze in einem Bytepuffer in einzelne Datenfelder;

CSV-Datei im Textmode oder Binärmode schreiben/lesen

Eine CSV-Datei kann im Textmode oder im Binärmode mit Hilfe der SPS-Funktionsbausteine für den Dateizugriff gelesen bzw. geschrieben werden. Abhängig von dem gewählten Modus ergeben sich Unterschiede mit Vor- und Nachteilen.

In 99% der Fälle können die CSV-Dateien im Textmode gelesen/geschrieben werden. Der Binärmode wird nur in den seltensten Fällen benötigt.

	Textmode	Binärmode
Funktionsbaustein für den Dateilesezugriff	**FB_FileGets** (Besonderheit: Das CR-Zeichen am Ende vom letzten Datensatz wird von diesem Baustein automatisch beim Lesezugriff aus der Datei entfernt. Damit der FB_CSVMemBufferReader Baustein einen solchen Datensatz interpretieren kann muss dieses Zeichen vorher wiederhergestellt/ eingefügt werden)	**FB_FileRead**
Funktionsbaustein für den Dateischreibzugriff	**FB_FilePuts** (Besonderheit: Ein zusätzliches CR-Zeichen am Ende vom letzten Datensatz wird von diesem Baustein automatisch beim Schreibzugriff in die Datei hinzugefügt. Der FB_CSVMemBufferWriter generiert aber die CR-Zeichen auch. Damit das Zeichen nicht doppelt in der CSV-Datei auftaucht muss dieses vor dem Schreibzugriff aus dem Puffer entfernt werden)	**FB_FileWrite**
Programmieraufwand	Kleiner	Größer
Sonderzeichen, nicht-druckbare Steuerzeichen im Datenfeld	Nicht erlaubt	Erlaubt
Maximale Datensatzlänge die geschrieben/gelesen werden kann	Auf 253 Zeichen begrenzt (Datensatz + CRLF). D.h. die Datensatzlänge darf 253 Zeichen nicht überschreiten .	Die maximale Datensatzlänge ist theoretisch unbegrenzt.
Ein kompletter Datensatz kann mit dem Baustein für den Schreibzugriff geschrieben werden	Ja	Ja
Ein kompletter Datensatz kann mit dem Baustein für den Lesezugriff gelesen werden	Ja Ein Datensatz in einer reinen Textdatei endet mit CRLF. CRLF markiert in einer solchen Datei das Zeilenende. Der FB_FileGets-Funktionsbaustein liest die Daten bis zum CRLF.	Nein
Binärdaten im Datenfeld	Nicht erlaubt	Erlaubt
Hilfsfunktionen für die Konvertierung der SPS-Daten in den CSV-Format und umgekehrt	CSVFIELD_TO_STRING [▶ 67] STRING_TO_CSVFIELD [▶ 70]	CSVFIELD_TO_ARG [▶ 69] ARG_TO_CSVFIELD [▶ 71]

	Textmode	Binärmode
Unterstützte SPS Variablentypen die direkt geschrieben/gelesen werden können	T_MaxString (STRING mit 255 Zeichen), andere Datentypen müssen zuerst in einen String konvertiert werden und dann als Datenfeld im Stringformat geschrieben/gelesen werden.	Beliebige Datentypen können geschrieben/gelesen werden
Beispielcode	P_TextModeRead() P_TextModeWrite()	P_BinaryModeRead() P_BinaryModeWrite()

Beispielprojekt

Das Beispielprojekt beinhaltet eigentlich 4 Beispiele: 2 für den Schreib-/Lesezugriff im Textmode (bevorzugt) und 2 für den Schreib-/Lesezugriff im Binärmode (selten):

P_TextModeRead();

P_TextModeWrite();

P_BinaryModeRead();

P_BinaryModeWrite();

Mit dem Beispielprojekt generierte CSV-Dateien:

Datenfelder ohne Binärdaten: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851054731.csv

Datenfelder beinhalten Binärdaten: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851056139.csv (bitte beachten Sie dass diese Datei nur von spezieller Software richtig interpretiert werden kann)

Prinzipieller Programmablauf beim Lesen einer CSV-Datei im Textmode:

1. Schritt: Die CSV-Datei im Textmode öffnen (FB_FileOpen). Wenn erfolgreich dann zu Schritt 2 gehen.

2. Schritt: Eine Zeile mit dem Funktionsbaustein FB_FileGets lesen. Das CR-Zeichen anhängen (siehe Hinweise in der Tabelle). Wenn erfolgreich dann zu Schritt 3 gehen, andernfalls zu Schritt 4 gehen (das Ende der Datei wurde erreicht oder ein Fehler ist aufgetreten).

3. Schritt: Die gelesene Zeile mit dem Funktionsbaustein FB_CSVMemBufferReader parsen. Es werden dabei die einzelnen Datenfelder gelesen. Danach zu Schritt 2 springen und die nächste Zeile lesen. Schritte 2 und 3 so lange wiederholen bis das Ende der Datei erreicht wurde oder ein Fehler aufgetreten ist.

4. Schritt: Die CSV-Datei schließen (FB_FileClose).

Prinzipieller Programmablauf beim Schreiben einer CSV-Datei im Textmode.

1. Schritt: Die CSV-Datei im Textmode öffnen (FB_FileOpen). Wenn erfolgreich dann zu Schritt 2 gehen.

2. Schritt: Mit dem Funktionsbaustein FB_CSVMemBufferWriter einen neuen Datensatz generieren. Die einzelnen Datenfelder werden dafür in einen Puffer hineingeschrieben. Dieser Puffer kann auch ein größerer String sein. Das CR-Zeichen vom Ende des Datensatzes entfernen und zu Schritt 3 gehen.

3. Schritt: Eine Zeile mit dem Funktionsbaustein FB_FilePuts schreiben. Dann Schritte 2 und 3 so lange wiederholen bis alle Datensätze geschrieben worden sind. Danach zu Schritt 4 gehen.

4. Schritt: Datei schließen (FB_FileClose).

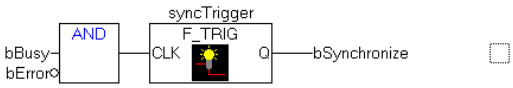
7.9 Beispiel: Software-Uhren (RTC, RTC_EX, RTC_EX2)

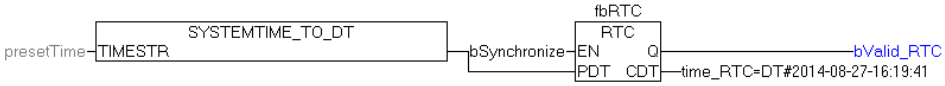
Die kompletten Sourcen finden Sie hier: https://infosys.beckhoff.com/content/1031/tcplclibutilities/Resources/11851057547.zip

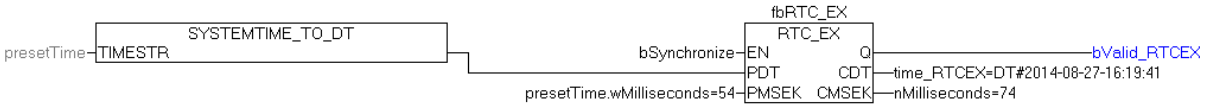
Im folgenden Beispiel werden die drei Software-Uhren alle 5 Sekunden mit der lokalen Windows-Systemzeit synchronisiert (die lokale Windows-Systemzeit wird in der Taskleiste eingeblendet).

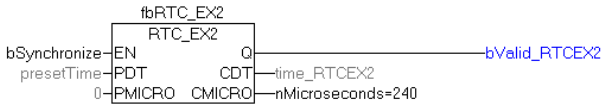
```
PROGRAM MAIN
VAR
    fbGetLocalTime  : NT_GetTime;
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    presetTime      : TIMESTRUCT;

    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 Anhang

8.1 Formatspezifikation

Diese Formatspezifikation wird von dem Funktionsbaustein FB_FormatString [▶ 156] und der Funktion F_FormatArgToStr [▶ 54] verwendet. Während bei dem Funktionsbaustein die Formatspezifikation über eine String-Eingangsvariable übergeben wird, geschieht es bei der Funktion über die einzelnen Funktionsparameter.

Die Formatspezifikation, welche die erforderlichen und optionalen Parameterfelder beinhaltet, hat folgende Form:

%[ ] [ ] [.] Flags [▶ 276] Width [▶ 277] Precision [▶ 277] Type [▶ 275]

Die einfachste Formatspezifikation beinhaltet nur das Prozentzeichen und das Typfeld (z.B. %s). Alle Zeichen, die dem Prozentzeichen folgen werden bis zu dem Typfeld als Parameterfelder ausgewertet. Zeichen vor dem Prozentzeichen und hinter dem Typfeld werden in den Ausgabestring kopiert. Beim Auftreten von nicht identifizierbaren oder unzulässigen Zeichen wird die Formatierung mit einem Fehler abgebrochen. Um das Prozentzeichen in dem Ausgabestring auszugeben benutzen Sie zwei nacheinander folgende Prozentzeichen (%%)

Type

Ein erforderliches Parameterfeld. Es ist ein ASCII-Zeichen (Character), welches bestimmt, ob das dazugehörige Argument als String, Integer oder Fließkommazahl interpretiert wird. Bitte beachten Sie dass einige der Typfeld-Parameter "case-sensitive" sind.

Typ	Argument	Ausgabe
b, B	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Binärstring (z.B.: '101010111000').
o, O	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Oktalstring.
u, U	BYTE, WORD, DWORD, **SINT, **INT, **DINT, USINT, UINT, UDINT	Dezimalstring ohne Vorzeichen.
c, C	BYTE, USINT	Einzelner (ASCII) Byte-Character.
f, F	***REAL, LREAL	Fließkommazahl. Der String hat die Form: **[–]dddd.dddd**, (dddd sind Dezimalzahlen). Die Anzahl der Zahlen vor dem Dezimalpunkt hängt von dem Wert der Fließkommazahl ab. Die Anzahl der Zahlen hinter dem Dezimalpunkt hängt von der geforderten Präzision ab. Ein Vorzeichen erscheint nur bei negativen Werten. Bei einem unendlichen positiven Wert wird **'#INF'** und bei unendlichem negativen Wert: **'-#INF'** zurückgeliefert. Wenn die übergebene Variable einen unzulässigen Wert besitzt (NaN, Not-a-Number), dann wird **'#QNAN'** oder **'-#QNAN'** zurückgeliefert. Wenn die Länge des formatierten Strings die maximal zulässige Länge des resultierenden Strings überschreitet, dann wird **'#OVF'** oder **'-#OVF'** zurückgeliefert.
d, D	BYTE, WORD, DWORD, SINT, INT, DINT, USINT, UINT, UDINT	Dezimalstring. Ein Vorzeichen erscheint nur bei negativen Werten.

Typ	Argument	Ausgabe
s, S	STRING	Single-byte–character string. Zeichen werden ausgegeben bis zur abschließenden Null oder bis der Parameter *precision* erreicht wurde.
X	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Hexadezimalstring. Für die Formatierung werden die (upper-case) Zeichen benutzt ('ABCDEF').
x	BYTE, WORD, DWORD, *REAL, **SINT, **INT, **DINT, USINT, UINT, UDINT	Hexadezimalstring. Für die Formatierung werden die (lower-case) Zeichen benutzt ('abcdef').
E	Not implemented. Reserved for future use!.	Fließkommazahlen in der wissenschaftlichen Notation.
e	Not implemented. Reserved for future use!.	Fließkommazahlen in der wissenschaftlichen Notation.

* Es wird der Speicherinhalt der REAL-Variable als Binärstring-, Oktalstring, Hexadezimalstring oder Dezimalstring ausgegeben.

** Es wird der Speicherinhalt der vorzeichenbehafteten Typen als Binärstring-, Oktalstring, Hexadezimalstring oder Dezimalstring ausgegeben.

*** Die REAL-Variable wird in den LREAL-Typ konvertiert und dann formatiert.

Flags

Optionales Parameterfeld. Es können eins oder mehrere Flags in einer beliebigen Reihenfolge bestimmt werden. Diese Parameter bestimmen die Ausrichtung des formatierten Wertes, Ausgabe von Vorzeichen, Leerzeichen und der Binär-/Oktal-/Hex-Prefixe.

Flag	Bedeutung	Type	Standard
-	Linksausrichtungs-Flag. Linksausrichtung des formatierten Wertes innerhalb der vorgegebenen Breite im Parameter: **Width**.	Kann in Verbindung mit allen Typen benutzt werden.	Rechtsausrichtung.
+	Vorzeichen-Flag. Erzwingt die Ausgabe des positiven Vorzeichens bei vorzeichenbehafteten positiven Zahlen.	Nur in Verbindung mit **e**, **E**, **f**, **F**, **d,D**, sonst wird das Flag ignoriert.	Das negative Vorzeichen erscheint nur bei negativen Werten.
0	Null-Flag. Wenn dieses Flag dem **Width**-Parameter vorangestellt wurde, wird der resultierende String von Links mit Nullen aufgefüllt bis die gewünschte Breite erreicht wurde.	Nur in Verbindung mit **e**, **E**, **f**, **F**, **s**, **S**, sonst wird das Flag ignoriert. Das Null-Flag wird auch ignoriert wenn zusätzlich auch das Linksausrichtungs-Flag (-) gesetzt wurde.	Kein Auffüllen mit Nullen.
Leerzeichen (' ')	Leerzeichen-Flag. Ein Leerzeichen wird einem positiven Wert vorangestellt	Nur in Verbindung mit **e**, **E**, **f**, **F**, **d**, **D**, sonst wird das Flag ignoriert. Das Leerzeichen-Flag wird auch ignoriert wenn gleichzeitig auch das Vorzeichen-Flag (**+**) gesetzt wurde	Kein Leerzeichen.
#	Prefix-Flag. Dem formatierten Wert wird ein "IEC" oder "Standard C" Prefix vorangestellt. "IEC"-Prefixe: **2#**, **8#**, **16# (default)** "Standard C"-Prefixe: **0**, **0x**, **0X**	Nur in Verbindung mit **b**, **B**, **o**, **O**, **x**, **X**, sonst wird das Flag ignoriert. Der "Standard-C"-Prefix-Typ kann durch das Setzen der globalen	Kein Prefix.

Flag	Bedeutung	Type	Standard
		Variablen GLOBAL_FORMAT_HASH_PREFIX_TYPE im Programm aktiviert werden: GLOBAL_FORMAT_HASH_PREFIX_TYPE := HASHPREFIX_STDC;	

Width

Optionales Parameterfeld. Dieser Parameter muss einen positiven dezimalen Wert haben. Er bestimmt die minimale Anzahl der ausgegebenen Zeichen im Ausgabestring. Es werden Leerzeichen (Links oder Rechts, abhängig von dem Ausrichtungs-Flag) dem Ausgabestring hinzugefügt, bis die gewünschte Breite erreicht wurde. Wenn das Null-Flag dem *Width*-Parameter vorangestellt wurde, wird der resultierende String von Links mit Nullen aufgefüllt, bis die gewünschte Breite erreicht wurde. Der Ausgabestring wird niemals durch den *Width*-Parameter auf die gewünschte Länge abgeschnitten!

Für den *Width*-Parameter kann auch ein Sternchen (*) angegeben werden. Der benötigte Wert wird dann von einem Argument geliefert (zulässige Typen: BYTE, WORD, DWORD, USINT, UINT, UDINT). Dem Argument für den *Width*-Parameter folgt dann das Argument für den zu formatierenden Wert.

Precision

Optionales Parameterfeld. Dieser Parameter folgt nach dem Punkt (.) und muss einen positiven dezimalen Wert besitzen. Folgt dem Punkt kein Wert, dann wird für die Präzision der Standardwert genommen (siehe Tabelle).

Typ	Bedeutung	Standard
b, B, o, O, u, U, x, X, d, D	Der Präzisionsparameter legt fest, wie viele Dezimalzeichen (digits) in dem Ausgabestring ausgegeben werden. Bei fehlenden Zeichen wird von Links mit Nullen aufgefüllt. Der Ausgabestring wird aber niemals abgeschnitten.	Standard: 1
c, C	Hat keine Bedeutung und wird ignoriert.	Ein Zeichen wird ausgegeben.
f, F	Der Präzisionsparameter legt die Anzahl der Nachkommastellen fest. Der Argument-Wert wird immer auf die entsprechende Anzahl der Nachkommastellen gerundet.	Standard: 6 Nachkommastellen
s, S	Der Präzisionsparameter legt fest, wie viele Zeichen aus dem Argument-String ausgegeben werden. Zeichen, die den Präzisionswert überschreiten werden nicht ausgegeben.	Es werden alle Zeichen bis zur abschließenden Null ausgegeben.

Für den *Precision*-Parameter kann auch ein Sternchen (*) angegeben werden. Der benötigte Wert wird dann von einem Argument geliefert (zulässige Typen: BYTE, WORD, DWORD, USINT, UINT, UDINT). Dem Argument für den *Precision*-Parameter folgt dann das Argument für den zu formatierenden Wert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 Build >1021	PC or CX (x86)	TcUtilities.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	

8.2 Schreiben der pers. Daten: Systemverhalten

Schreibtrigger	Interne Optimierungen bei der Verwaltung der persistenten Daten	Datenkonsistenz der persistenten Daten	Mögliche Zyklusüberschreitung beim Schreiben
Funktionsbaustein WritePersistentData [▶ 218]	Keine	Alle Daten sind aus einem Zyklus	Ja, wenn das Schreiben aller Daten länger dauert als ca. der SPS-Zyklus
Funktionsbaustein FB_WritePersistentData [▶ 151] mit SPDM_2PASS [▶ 235]	Ja	Alle Daten sind aus einem Zyklus	Ja, wenn das Schreiben aller Daten länger dauert als ca. der SPS-Zyklus
Funktionsbaustein FB_WritePersistentData [▶ 151] mit SPDM_VAR_BOOST [▶ 235]	Ja	Daten einzelner persistenten Variablen sind aus einem Zyklus	Selten, wenn das Schreiben der größten persistenten Daten länger dauert als ca. der SPS-Zyklus)
TwinCAT-System-Stopp (die persistenten Daten werden beim Stopp automatisch geschrieben)	Ja	Alle Daten sind aus einem Zyklus	Keine

Mehr Informationen:
www.beckhoff.com/tx1200

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