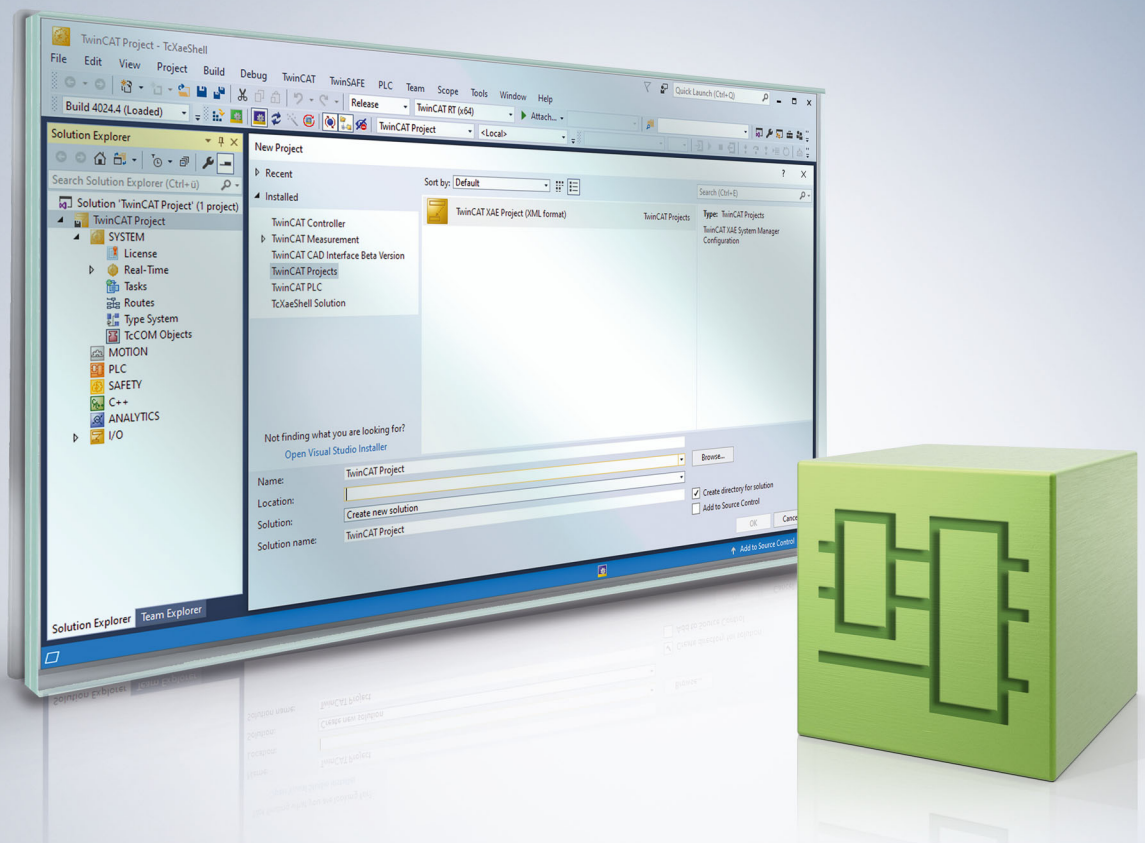
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC-Bibliothek: Tc2_System



10.06.2025 | Version: 1.16.5

Inhaltsverzeichnis

1 Vorwort ... 7
1.1 Hinweise zur Dokumentation ... 7
1.2 Zu Ihrer Sicherheit ... 8
1.3 Hinweise zur Informationssicherheit ... 9
2 Übersicht ... 10
3 Funktionsbausteine ... 14
3.1 Generelle Funktionsbausteine ... 14
3.1.1 DRAND ... 14
3.1.2 FB_IecCriticalSection ... 14
3.1.3 FB_ReadTaskExceedCounter ... 16
3.1.4 FB_ResetTaskExceedCounter ... 17
3.1.5 FB_SetLedColor_BAPI ... 17
3.1.6 FB_SetLedColorEx_BAPI ... 18
3.1.7 GETCURTASKINDEX ... 19
3.1.8 FB_CreateGUID ... 20
3.2 ADS-Funktionsbausteine ... 21
3.2.1 Control + Status ... 21
3.2.2 Indication + Response ... 26
3.2.3 ADSREAD ... 40
3.2.4 ADSREADEX ... 41
3.2.5 ADSWRITE ... 43
3.2.6 ADSRDWRT ... 44
3.2.7 ADSRDWRTEX ... 46
3.3 Datei-Funktionsbausteine ... 48
3.3.1 FB_EOF ... 48
3.3.2 FB_FileOpen ... 49
3.3.3 FB_FileClose ... 52
3.3.4 FB_FileLoad ... 54
3.3.5 FB_FileGets ... 55
3.3.6 FB_FilePuts ... 57
3.3.7 FB_FileRead ... 59
3.3.8 FB_FileWrite ... 61
3.3.9 FB_FileSeek ... 63
3.3.10 FB_FileTell ... 64
3.3.11 FB_FileDelete ... 66
3.3.12 FB_FileRename ... 67
3.3.13 FB_CreateDir ... 69
3.3.14 FB_RemoveDir ... 71
3.4 EventLogger-Funktionsbausteine ... 72
3.4.1 ADSLOGEVENT ... 72
3.4.2 ADSCLEAREVENTS ... 75
3.4.3 FB_SimpleAdsLogEvent ... 76
3.5 IEC-Schritt-/ SFC-Flags-Funktionsbausteine ... 78
3.5.1 AnalyzeExpression ... 78

3.5.2 AnalyzeExpressionTable ... 81
3.5.3 AnalyzeExpressionCombined ... 83
3.5.4 AppendErrorString ... 83
3.5.5 SFCActionControl ... 84
3.6 Watchdog-Funktionsbausteine ... 84
3.6.1 FB_PcWatchdog ... 84
3.6.2 FB_PcWatchDog_BAPI ... 86
3.7 Zeitfunktionsbausteine ... 88
3.7.1 GETCPUACCOUNT ... 88
3.7.2 GETCPUCOUNTER ... 88

4 Funktionen ... 90
4.1 Generelle Funktionen ... 90
4.1.1 F_CheckMemoryArea ... 90
4.1.2 F_CmpLibVersion ... 91
4.1.3 F_CreateIPv4Addr ... 91
4.1.4 F_ScanIPv4AddrIds ... 92
4.1.5 F_GetCpuCoreIndex ... 93
4.1.6 F_GetCpuCoreInfo ... 93
4.1.7 F_GetMappingPartner ... 94
4.1.8 F_GetMappingStatus ... 94
4.1.9 F_GetStructMemberAlignment ... 95
4.1.10 F_GetTaskInfo ... 98
4.1.11 F_RaiseException ... 99
4.1.12 F_SplitPathName ... 100
4.1.13 SETBIT32 ... 102
4.1.14 CSETBIT32 ... 102
4.1.15 GETBIT32 ... 103
4.1.16 CLEARBIT32 ... 104
4.1.17 GETCURTASKINDEXEX ... 104
4.1.18 LPTSIGNAL ... 105
4.1.19 TestAndSet ... 106
4.2 ADS-Funktionen ... 107
4.2.1 ADSLOGDINT ... 107
4.2.2 ADSLOGLREAL ... 109
4.2.3 ADSLOGSTR ... 110
4.2.4 F_CreateAmsNetId ... 112
4.2.5 F_ScanAmsNetIds ... 113
4.3 Character-Funktionen ... 113
4.3.1 F_ToCHR ... 113
4.3.2 F_ToASC ... 114
4.4 I/O-Portzugriff ... 115
4.4.1 F_IOPortRead ... 115
4.4.2 F_IOPortWrite ... 115
4.5 Memory-Funktionen ... 117
4.5.1 MEMCMP ... 117
4.5.2 MEMCPY ... 118

4.5.3 MEMMOVE ... 119
4.5.4 MEMSET ... 121
4.6 Zeit-Funktionen ... 122
4.6.1 F_GetSystemTime ... 122
4.6.2 F_GetTaskTime ... 122
4.6.3 F_GetTaskTotalTime ... 123
4.7 [veraltet] ... 123
4.7.1 F_GetVersionTcSystem ... 123
4.7.2 GETSYSTEMTIME ... 124
4.7.3 GETTASKTIME ... 125

5 Datentypen ... 126
5.1 E_IOAccessSize ... 126
5.2 E_OpenPath ... 126
5.3 E_SeekOrigin ... 126
5.4 E_TcEventClass ... 127
5.5 E_TcEventClearModes ... 127
5.6 E_TcEventPriority ... 127
5.7 E_TcEventStreamType ... 128
5.8 E_TcMemoryArea ... 128
5.9 E_UsrLED_Color ... 128
5.10 EPlcMappingStatus ... 129
5.11 ST_AmsAddr ... 129
5.12 ST_CpuCoreInfo ... 129
5.13 SYSTEMINFOTYPE ... 130
5.14 SYSTEMTASKINFOTYPE ... 130
5.15 T_AmsNetID ... 130
5.16 T_AmsNetIdArr ... 130
5.17 T_AmsPort ... 130
5.18 T_IPv4Addr ... 132
5.19 T_IPv4AddrArr ... 132
5.20 T_MaxString ... 132
5.21 TcEvent ... 133

6 Globale Konstanten ... 135
6.1 Konstanten ... 135
6.2 Bibliotheksversion ... 140

7 Beispiele ... 141
7.1 Beispiel mit AdsReadInd-/AdsReadRes-Funktionsbausteinen ... 141
7.2 Beispiel mit AdsWriteInd-/AdsWriteRes-Funktionsbausteinen ... 143
7.3 Beispiel mit AdsRead-Funktionsbaustein ... 144
7.4 Beispiel mit AdsWrite-Funktionsbaustein ... 145
7.5 EventLogger-Meldungen aus der SPS senden/quittieren ... 146
7.6 Dateizugriff aus der SPS ... 147
7.7 Testen der CPU Reserve eines CX70xx ... 150

8 Anhang ... 152
8.1 ADS Return Codes ... 152

8.2 Support und Service 157

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®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® und XTS® 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 Kennzeichnungen führen.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

Fremdmarken

In dieser Dokumentation können Marken Dritter verwendet werden. Die zugehörigen Markenvermerke finden Sie unter: https://www.beckhoff.com/trademarks.

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

Nicht alle Funktionsbausteine und Funktionen, die häufig in SPS-Applikationen benötigt werden, sind in der IEC61131-3 genormt. Die Bibliothek Tc2_System enthält solche Funktionen und Funktionsbausteine des TwinCAT-Systems, die nicht zum Standardumfang der IEC61131-3 gehören und dementsprechend herstellerspezifisch sind.

Generelle Funktionsbausteine

Name	Beschreibung
DRAND [▶ 14]	Zufallszahlengenerator
FB_lecCriticalSection [▶ 14]	Kritische Bereiche unter gegenseitigen Ausschluss stellen
FB_SetLedColor_BAPI [▶ 17]	Schaltet User-LED auf PCs und Embedded PCs mit BIOS-API-Unterstützung
GETCURTASKINDEX [▶ 19]	Ermittelt den aktuellen Taskindex

ADS-Funktionsbausteine

Name	Beschreibung
ADSREAD [▶ 40]	Lesen von Daten über ADS
ADSREADEX [▶ 41]	Lesen von Daten über ADS und Ermittlung der tatsächlichen Anzahl der gelesenen Bytes
ADSWRITE [▶ 43]	Schreiben von Daten über ADS
ADSRDWRT [▶ 44]	Schreiben und Lesen von Daten über ADS
ADSRDWRTEX [▶ 46]	Schreiben und Lesen von Daten über ADS und Ermittlung der tatsächlichen Anzahl der gelesenen Bytes
ADSRDSTATE [▶ 21]	Lesen des Status eines Gerätes über ADS
ADSWRTCTL [▶ 23]	Schreiben/Setzen des Status eines Gerätes über ADS
ADSRDDEVINFO [▶ 25]	Lesen der Geräteinformation über ADS

Erweiterte ADS-Funktionsbausteine

Name	Beschreibung
ADSREADIND [▶ 28]	ADSREAD-Indication
ADSWRITEIND [▶ 31]	ADSWRITE-Indication
ADSRDWRTIND [▶ 34]	ADSRDWRT-Indication
ADSREADRES [▶ 37]	ADSREAD-Response
ADSWRITERES [▶ 38]	ADSWRITE-Response
ADSRDWRTRES [▶ 39]	ADSRDWRT-Response

Funktionsbausteine für Dateizugriffe

Mit den nachfolgend aufgeführten Funktionsbausteinen können lokal auf dem PC Dateien aus der SPS heraus bearbeitet werden. Durch die AMS-Netzwerkadresse wird das TwinCAT-Zielsystem identifiziert. Durch diesen Mechanismus ist es unter anderem möglich, Dateien auf anderen TwinCAT-Systemen des Verbundes anzulegen bzw. zu bearbeiten. Der Zugriff auf Dateien besteht aus drei aufeinanderfolgenden Phasen:

1. Öffnen der Datei
2. Lesender oder schreibender Zugriff auf die geöffnete Datei
3. Schließen der Datei

Das Öffnen der Datei dient dazu, eine temporäre Verbindung zwischen der externen Datei, von der zunächst nur der Name bekannt ist, und dem laufenden Programm herzustellen. Das Schließen der Datei dient dazu, das Ende der Bearbeitung anzuzeigen und sie in einen definierten Ausgangszustand für die Bearbeitung durch andere Programme zu versetzen.

Name	Beschreibung
FB_EOF [▶ 48]	Testen auf Dateiende
FB_FileOpen [▶ 49]	Öffnen einer Datei
FB_FileClose [▶ 52]	Schließen einer Datei
FB_FileGets [▶ 55]	String aus einer Text-Datei lesen
FB_FilePuts [▶ 57]	Nullterminierten-String in eine Text-Datei schreiben
FB_FileRead [▶ 59]	Lesen aus einer Datei
FB_FileWrite [▶ 61]	Schreiben in eine Datei
FB_FileSeek [▶ 63]	Verstellen des Dateizeigers
FB_FileTell [▶ 64]	Ermitteln der aktuellen Position des Dateizeigers
FB_FileDelete [▶ 66]	Löschen einer Datei
FB_FileRename [▶ 67]	Umbenennen einer Datei
FB_CreateDir [▶ 69]	Erstellen eines neuen Verzeichnisses
FB_RemoveDir [▶ 71]	Löschen eines Verzeichnisses

EventLogger-Funktionsbausteine

Der TwinCAT EventLogger hat die Aufgabe, alle auftretenden Meldungen (Events) im TwinCAT-System zu verwalten, weiterzuleiten und gegebenenfalls in die TwinCAT-Logdatei zu schreiben.

Name	Beschreibung
ADSLOGEVENT [▶ 72]	Absenden und Quittieren von Meldungen zum TwinCAT EventLogger.
ADSCLEAREVENTS [▶ 75]	Absenden und Quittieren von Meldungen zum TwinCAT EventLogger.
FB_SimpleAdsLogEvent [▶ 76]	Absenden und Quittieren von Meldungen zum TwinCAT EventLogger.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

Der TwinCAT EventLogger wurde durch den Nachfolger TwinCAT 3 EventLogger abgelöst. Der ältere TwinCAT EventLogger wird von TwinCAT 3 bis zur Version 3.1.4024 unterstützt. Neuere TwinCAT-Versionen (>= 3.1.4026.0) unterstützen nur den neueren TwinCAT 3 EventLogger. SPS-Funktionsbausteine hierzu befinden sich in der SPS Bibliothek Tc3_EventLogger.

IEC-Schritt- / SFC-Flags-Funktionsbausteine

Diese Funktionen/Funktionsbausteine werden benötigt, wenn in SFC-Programmen/Projekten die IEC-Schritte oder SFC-Flags benutzt werden.

Name	Beschreibung
AnalyzeExpression [▶ 78]	Wird bei der Benutzung der SFC-Flags benötigt
AnalyzeEspressionTable [▶ 81]	Wird bei der Benutzung der SFC-Flags benötigt
AnalyzeExpressionCombined [▶ 83]	Wird bei der Benutzung der SFC-Flags benötigt
AppendErrorString [▶ 83]	Wird bei der Benutzung der SFC-Flags benötigt, um Strings mit Fehlerbeschreibung zu formatieren.
SFCActionControl [▶ 84]	Ermöglicht die Benutzung der IEC-Schritte

Watchdog-Funktionsbausteine

Name	Beschreibung
FB_PcWatchdog [▶ 84]	Aktiviert oder deaktiviert den PC-Watchdog Ist nur verfügbar auf IPCs mit den Mainboards IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051
FB_PcWatchdog_BAPI [▶ 86]	Aktiviert oder deaktiviert den PC-Watchdog Ist nur verfügbar auf IPCs mit den Mainboards CBxx63 mit einer BIOS-Version >=0.44

Zeitfunktionsbausteine

Name	Beschreibung
GETCPUCOUNTER [▶ 88]	Zyklusticker der CPU auslesen
GETCPUACCOUNT [▶ 88]	Zyklusticker der SPS-Task auslesen

Generelle Funktionen

Name	Beschreibung
F_CheckMemoryArea [▶ 90]	Liefert Informationen darüber, in welchem Speicherbereich sich die angefragte Variable mit angegebener Größer befindet
F_CmpLibVersion [▶ 91]	Vergleicht eine existierende Bibliothek mit der Version, die benötigt wird
F_CreateIPv4Addr [▶ 91]	Konvertiert einzelne IPv4-Adressbytes in einen String
F_ScanIPv4AddrIds [▶ 92]	Konvertiert IPv4-Adressstring in einzelne Adressbytes
F_GetMappingPartner [▶ 94]	Liefert die Objekt- ID der Partnerseite des Mappings
F_GetMappingStatus [▶ 94]	Liefert den aktuellen Mapping- Status einer SPS-Variablen
F_GetStructMemberAlignment [▶ 95]	Liest Informationen über die verwendete Speicherausrichtung (alignment)
F_SplitPathName [▶ 100]	Zerlegt den Pfadnamen in vier Einzelkomponenten
SETBIT32 [▶ 102]	Setzen eines Bits in einem DWORD
CSETBIT32 [▶ 102]	Setzen/Zurücksetzen eines Bits in einem DWORD
GETBIT32 [▶ 103]	Lesen eines Bits aus einem DWORD
CLEARBIT32 [▶ 104]	Löschen eines Bits in einem DWORD
GETCURTASTINDEXEX [▶ 104]	Ermitteln des Taskindex
LPTSIGNAL [▶ 105]	Ausgabe eines Signals auf einem Pin des parallelen Port
TestAndSet [▶ 106]	Flag setzen und prüfen, ohne dass dies unterbrochen werden kann

ADS-Funktionen

Funktionen, die mithilfe der ADS-Schnittstelle Teilfunktionalitäten des Windows-NT-Betriebssystems (wie z. B. die Ausgabe von Message-Boxen) durch Aufrufmöglichkeiten in der SPS zugänglich machen.

Name	Beschreibung
ADSLOGDINT [▶ 107]	Ausgabe einer DINT Variablen ins Logbuch und/oder Messagebox
ADSLOGLREAL [▶ 109]	Ausgabe einer (L)REAL Variablen ins Logbuch und/oder Messagebox
ADSLOGSTR [▶ 110]	Ausgabe einer STRING Variablen ins Logbuch und/oder Messagebox
F_CreateAmsNetId [▶ 112]	Generiert einen formatierten AmsNetId-String
F_ScanAmsNetIds [▶ 113]	Konvertiert einen AmsNetId-String in einzelnen Adressbytes

Charakter-Funktionen

Name	Beschreibung
F_ToCHR [▶ 113]	Konvertiert ASCII-Code in ein Charakterzeichen
F_ToASC [▶ 114]	Konvertiert Charakterzeichen in den ASCII-Code

I/O-Portzugriff

Name	Beschreibung
F_IOPortRead [▶ 115]	Liest I/O Port
F_IOPortWrite [▶ 115]	Beschreibt I/O Port

Memory-Funktionen

Funktionen für den direkten Zugriff auf Speicherbereiche des SPS-Laufzeitsystems.

HINWEIS

Systemabsturz oder Zugriff auf unerlaubte Speicherbereiche

Da mit den Funktionen direkt auf den physikalischen Speicher zugegriffen wird, ist bei deren Anwendung besondere Vorsicht geboten. Falsche Parameterwerte können zu einem Systemabsturz oder einem Zugriff auf unerlaubte Speicherbereiche führen.

Name	Beschreibung
MEMCMP [▶ 117]	Werte der Variablen in zwei Speicherbereichen vergleichen
MEMCPY [▶ 118]	Variablenwerte von einem Speicherbereich in einen anderen kopieren
MEMMOVE [▶ 119]	Variablen von überlappenden Speicherbereichen kopieren
MEMSET [▶ 121]	Variablen in einem Speicherbereich auf einen bestimmten Wert setzen

Zeit-Funktionen

Name	Beschreibung
F_GetSystemTime [▶ 122]	Betriebssystem-Zeitstempel auslesen
F_GetTaskTime [▶ 122]	(Soll-)Startzeit der Task auslesen

3 Funktionsbausteine

3.1 Generelle Funktionsbausteine

3.1.1 DRAND

Der Funktionsbaustein DRAND erlaubt die Erzeugung einer (Pseudo-) Zufallszahl des Typs LREAL.

Eingänge

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

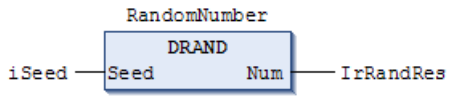
Name	Typ	Beschreibung
Seed	INT	Initialwert zur Festlegung der Zufallszahlenreihe.

Ausgänge

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

Name	Typ	Beschreibung
Num	LREAL	Dieser Ausgang gibt eine Pseudo-Zufallszahl im Bereich 0.0 .. 1.0 mit doppelter Genauigkeit zurück. Der Generator erzeugt dabei eine Zahlenfolge mit 1075 stochastischen Werten je Periode.

Beispiel für den Aufruf des Bausteins in FBD:



Im Beispiel wird der LREAL-Wert 0.643412 erzeugt und zurückgegeben. Mit dem Eingangsparameter Seed kann der Initialwert der Reihe beeinflusst werden. Wenn beispielsweise eine deterministisch reproduzierbare Zufallszahlenfolge in verschiedenen Sitzungen erreicht werden soll, muss ein identischer Seed-Wert verwendet werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.1.2 FB_IecCriticalSection

Mit dem Funktionsbaustein werden kritische Bereiche unter gegenseitigen Ausschluss gestellt. Kritische Bereiche sind Modifikationen auf einer oder meist mehreren Variablen, die während der Änderungen einen inkonsistenten Zustand haben. Es ist daher zwingend erforderlich, dass solche Modifikationen nur von exakt einer Task zu einem Zeitpunkt durchgeführt wird. Der Baustein stellt dazu die Methoden Enter() und Leave() bereit. Durch den erfolgreichen Aufruf von Enter() wird der kritische Bereich betreten und gilt dann als belegt. Nach Abschluss der Modifikationen muss der kritische Bereich durch Leave() wieder verlassen werden.

Zykluszeitüberschreitung durch angehaltene Task

Wenn eine weitere Task durch Aufruf von Enter() einen bereits belegten kritischen Bereich betreten will, wird sie durch den TwinCAT Scheduler blockiert. Die Task wird angehalten, bis der Bereich wieder freigegeben ist! Nach der Freigabe wird die Bearbeitung des Programmcode fortgesetzt und der kritische Bereich betreten.

- Achten Sie darauf, dass die kritischen Bereiche sehr kurz gehalten werden, damit es bei der wartenden Task keine Zyklusüberschreitungen gibt. Mehrere wartende Tasks werden entsprechend ihrer Priorität in den kritischen Bereich gelassen.

Wenn eine Task durch den TwinCAT Scheduler blockiert wird, weil sie einen bereits belegten kritischen Bereich betreten wollte, geschieht dies ohne „Busy-Waiting“. Niederpriore Tasks können also die CPU Kapazität währenddessen nutzen.

Windows CE

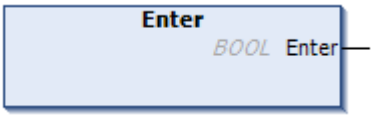
Die Funktionalität wird unter Windows-CE-Betriebssystemen ab TwinCAT v3.1.4022.29 unterstützt. (Bei einer älteren TwinCAT Version geben die Methoden FALSE aus.)

Alternative

Kritische Bereiche können auch mit der Funktion TestAndSet() [▶ 106] realisiert werden. Mit der Funktion kann die Belegung eines kritischen Bereiches markiert und geprüft werden. Die Funktion blockiert jedoch nicht und es besteht die Möglichkeit, dass der Bereich in einem Zyklus nicht durchlaufen werden kann.

Grundsätzlich muss die Anzahl der kritischen Bereiche möglichst klein und die Länge der kritischen Bereiche kurz gehalten werden.

Methode Enter()



Die Methode markiert den Anfang eines kritischen Bereiches.

Mögliche Rückgabewerte:

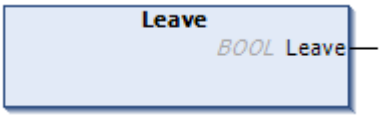
TRUE:

- Der kritische Bereich darf betreten werden.

FALSE:

- Der kritische Bereich darf nicht betreten werden.
- Der Baustein wird von der Runtime noch nicht unterstützt.
- Der kritische Bereich ist durch eine andere PLC-Task belegt. Diese Task steht in einem Breakpunkt im Stopp. Durch den Rückgabewert FALSE wird ein dauerhaftes Blockieren der Task vermieden und das Update der E/A gewährleistet.

Methode Leave()



Die Methode markiert das Ende eines kritischen Bereiches. Sie muss immer bei Abschluss des kritischen Bereiches aufgerufen werden.

Mögliche Rückgabewerte:

TRUE:

- Der Bereich wurde erfolgreich verlassen.

FALSE:

- Der Baustein wird von der Runtime nicht unterstützt.
- Der kritische Bereich war nicht mit Enter belegt worden.

Beispiel für die Verwendung des Bausteins:

Der Funktionsbaustein FB_IecCriticalSection bietet die Möglichkeit den Zugriff auf gemeinsame Daten abzusichern. Die Instanz des Funktionsbausteines wird hierzu ebenso wie die zu sichernden Daten global angelegt.

```
VAR_GLOBAL
    fbCrititcalSection : FB_IecCriticalSection;
END_VAR

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4020	PC oder CX (x86, x64) WES, WES7, Win7, Win10	Tc2_System (System)
TwinCAT v3.1.4022.29	PC oder CX (x86, ARM) WinCE	Tc2_System (System)

3.1.3 FB_ReadTaskExceedCounter

Der Funktionsbaustein liest den Exceed Counter aus. Der Exceed Counter wird immer dann vom System hochgezählt, wenn die ausgewählte Task die eingestellte Task-Zeit überschreitet. Das bedeutet, dass die Echtzeit in dem Zyklus nicht eingehalten werden konnte.

Die Gründe für das Überschreiten der Echtzeit können sehr vielfältig sein, aber in der Regel liegt es an der SPS-Laufzeit und der Applikation innerhalb dieser Laufzeit. Ein Beispiel hierfür sind Programmierschleifen wie FOR, WHILE, REPEAT, da diese immer in einem Zyklus bearbeitet werden.

Eingänge

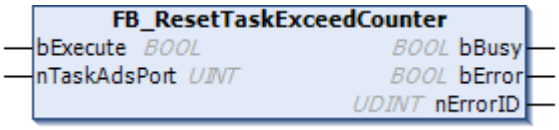
Name	Type	Beschreibung
bExecute	BOOL	Positive Flanke aktiviert den Baustein.
nTaskAdsPort	UINT	ADS Port der ausgewählten Task Beispiel einer möglichen Zuweisung: TwinCAT_SystemInfoVarList._TaskInfo[GETCURTASKINDEXEX()].AdsPort

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	Baustein ist aktiv und arbeitet.
bError	BOOL	Baustein hat einen Fehler.
nErrorID	UDINT	ADS Fehler Code
nExceedCounter	UDINT	Gelesener Wert des Exceed Counters

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.22	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.25.0

3.1.4 FB_ResetTaskExceedCounter



Der Funktionsbaustein kann den Exceed Counter zurücksetzen. Der Exceed Counter wird immer dann hochgezählt, wenn die ausgewählte Task die eingestellte Task-Zeit überschreitet. Das bedeutet, dass die Echtzeit in dem Zyklus nicht eingehalten werden konnte.

Die Gründe für das Überschreiten der Echtzeit können sehr vielfältig sein, aber in der Regel liegt es an der SPS-Laufzeit und der Applikation innerhalb dieser Laufzeit. Ein Beispiel hierfür sind Programmierschleifen wie FOR, WHILE, REPEAT, da diese immer in einem Zyklus bearbeitet werden.

Eingänge

Name	Type	Beschreibung
bExecute	BOOL	Positive Flanke aktiviert den Baustein.
nTaskAdsPort	UINT	ADS Port der ausgewählten Task. Beispiel einer möglichen Zuweisung: TwinCAT_SystemInfoVarList._TaskInfo[GETCURTASKINDEXEX()].AdsPort

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	Baustein ist aktiv und arbeitet.
bError	BOOL	Baustein hat einen Fehler.
nErrorID	UDINT	ADS Fehler Code

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.22	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.25.0

3.1.5 FB_SetLedColor_BAPI

Diese Funktionalität ist nur auf IPCs und Embedded PCs mit einer Usr-LED und mit einer BIOS-Version, die die BIOS-API unterstützt, verfügbar.

Mit dem Funktionsbaustein FB_SetLedColor_BAPI kann die User-LED auf PCs und embedded PCs mit BIOS-API-Unterstützung geschaltet werden. Die LED-Farbe wird über eine steigende Flanke an bExecute und die Farbe eNewColor geschaltet. Die LED kann ausgeschaltet werden (eNewColor = eUsrLED_Off) oder rot (eNewColor = eUsrLED_Red), blau (eNewColor = eUsrLED_Blue) oder grün (eNewColor = eUsrLED_Green) leuchten.

Eingänge

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

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Geräts (leerer String oder lokale Netzwerkkennung) (Typ T_AmsNetID [▶ 130])
eNewColor	E_UsrLED_Color	Neue LED-Farbe (Typ E_UsrLED_Color [▶ 128])
bExecute	BOOL	Mit steigender Flanke wird der Befehl ausgeführt. Der Eingang muss zurückgesetzt werden, sobald der Baustein nicht mehr aktiv ist (`bBusy=FALSE`).
tTimeout	TIME	Zeit bis zum Abbruch der internen ADS-Kommunikation.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, solange der Baustein aktiv ist.
bError	BOOL	TRUE, wenn bei der Ausführung des Befehls ein Fehler aufgetreten ist.
nErrID	UDINT	Enthält den ADS-Fehlercode oder den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Er wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_System (System) v3.4.14

3.1.6 FB_SetLedColorEx_BAPI

Diese Funktionalität ist nur auf IPCs und Embedded PCs mit einer Usr-LED und mit einer BIOS-Version, die die BIOS-API unterstützt, verfügbar.

Mit dem Funktionsbaustein FB_SetLedColorEx_BAPI können die User-LEDs (USR, U1 bzw. U2) auf PCs und Embedded PCs mit BIOS-API-Unterstützung geschaltet werden. Die LED-Farbe wird über eine steigende Flanke an bExecute und die Farbe eNewColor geschaltet. Die LED kann ausgeschaltet werden (eNewColor = eUsrLED_Off) oder rot (eNewColor = eUsrLED_Red), blau (eNewColor = eUsrLED_Blue) oder grün (eNewColor = eUsrLED_Green) leuchten.

Eingänge

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

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Geräts (leerer String oder lokale Netzwerkkennung) (Typ T_AmsNetID [▶ 130])
nLedID	USINT	ID für die Auswahl der User-LED: Für Geräte mit nur einer USR-LED wird über nLedID = 0 die USR-LED gewählt (Standard-Wert ist 0). Für Geräte mit mehreren User-LEDs wird über nLedID = 1 die U1-LED bzw. über nLedID = 2 die U2-LED gewählt.
eNewColor	E_UsrLED_Color	Neue LED-Farbe (Typ E_UsrLED_Color [▶ 128])
bExecute	BOOL	Mit steigender Flanke wird der Befehl ausgeführt. Der Eingang muss zurückgesetzt werden, sobald der Baustein nicht mehr aktiv ist (`bBusy=FALSE`).
tTimeout	TIME	Zeit bis zum Abbruch der internen ADS-Kommunikation.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, solange der Baustein aktiv ist.
bError	BOOL	TRUE, wenn bei der Ausführung des Befehls ein Fehler aufgetreten ist.
nErrID	UDINT	Enthält den ADS-Fehlercode oder den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Er wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_System (System) v3.6.1

3.1.7 GETCURTASKINDEX

Veralteter Funktionsbaustein

Dieser Funktionsbaustein ist veraltet. Verwenden Sie stattdessen die Funktion GETCURTASKINDEXEX() [▶ 104].

Der Funktionsbaustein GETCURTASKINDEX ermittelt den Taskindex der Task, in welcher er aktuell aufgerufen wird.

Um zu unterscheiden, ob der aktuelle Aufruf im Echtzeitkontext und von einer zyklischen SPS-Task erfolgt, beachten Sie die Dokumentation der Funktion GETCURTASKINDEXEX [▶ 104]. Beispielsweise erfolgt der automatische Aufruf von FB_init-Methoden während der Initialisierung nicht von einer zyklischen SPS-Task.

Eingänge

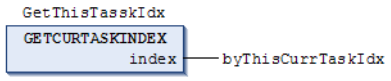
```
VAR_INPUT
(*none*)
END_VAR
```

Ausgänge

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

Name	Typ	Beschreibung
index	BYTE	Gibt den aktuellen Taskindex der aufrufenden Task zurück (1..4).

Beispiel für den Aufruf des Bausteins in FBD:



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.1.8 FB_CreateGUID

Der Funktionsbaustein erzeugt eine neue GUID. Es ist möglich eine Liste von unterschiedlichen neuen GUIDs mit einem Aufruf zu erhalten, wenn ein Array vom Typ GUID am Eingang als Puffer angegeben wird.

Eingänge

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

Name	Typ	Beschreibung
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.
pGuidBuffer	POINTER TO GUID	Gibt die Adresse auf den Puffer für erzeugte GUIDs an. Es ist möglich die Adresse auf ein ARRAY OF GUID anzugeben.
nGuidBufferSize	UDINT	Gibt die Größe in Bytes des angegebenen Puffers an.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrorId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152].

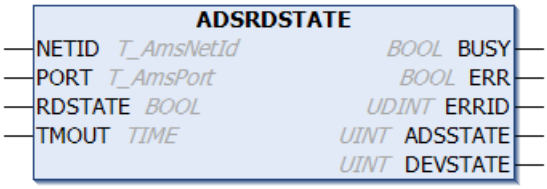
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4022	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.18.0

3.2 ADS-Funktionsbausteine

3.2.1 Control + Status

3.2.1.1 ADSRDSTATE



Der Funktionsbaustein fordert den Zustand eines ADS-Gerätes an.

Eingänge

```
VAR_INPUT
    NETID   : T_AmsNetId;
    PORT    : T_AmsPort;
```

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130]).
RDSTATE	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

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

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.
ADSSTATE	UDINT	Enthält die Zustandskennzahl des ADS-Zielgerätes.
DEVSTATE	UDINT	Spezifische Zustandskennzahl des ADS-Zielgerätes. Die hier zurück gelieferten Codes sind Zusatzinformationen, die für das ADS-Gerät spezifisch sind.

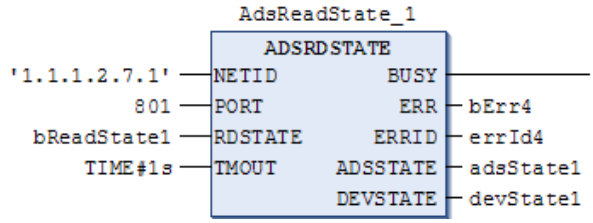
Zustandskennzahl des ADS-Zielgerätes

Die hier zurück gelieferten Codes sind festgelegt für alle ADS-Server:

- ADSSTATE_INVALID = 0;
- ADSSTATE_IDLE = 1;
- ADSSTATE_RESET = 2;
- ADSSTATE_INIT = 3;
- ADSSTATE_START = 4;
- ADSSTATE_RUN = 5;
- ADSSTATE_STOP = 6;
- ADSSTATE_SAVECFG = 7;
- ADSSTATE_LOADCFG = 8;
- ADSSTATE_POWERFAILURE = 9;
- ADSSTATE_POWERGOOD = 10;

- ADSSTATE_ERROR = 11;
- ADSSTATE_SHUTDOWN = 12;
- ADSSTATE_SUSPEND = 13;
- ADSSTATE_RESUME = 14;
- ADSSTATE_CONFIG = 15;
- ADSSTATE_RECONFIG = 16;
- ADSSTATE_STOPPING = 17;
- ADSSTATE_INCOMPATIBLE = 18;
- ADSSTATE_EXCEPTION = 19;

Beispiel für den Aufruf des Bausteins in FBD:

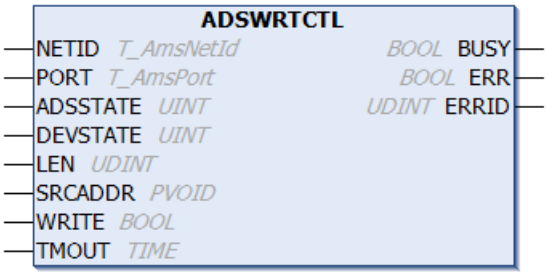


In dem Beispiel wird das SPS-Laufzeitsystem 1 (PortNr.801) auf dem Rechner mit der Netzwerkadresse 1.1.1.2.7.1 nach seinem Zustand gefragt. Die Antwort ist adsState = 1 (IDLE) ohne Zusatzcode devState=0.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.1.2 ADSWRTCTL



Der Funktionsbaustein erlaubt die Ausführung eines ADS-Kontrollbefehls, um den Zustand eines ADS-Gerätes zu beeinflussen, z. B. um ein Gerät zu starten, stoppen oder zurückzusetzen.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130]).
ADSSTATE	UINT	Zustandskennzahl des ADS-Zielgerätes.
DEVSTATE	UINT	Spezifische Zustandskennzahl des ADS-Zielgerätes. Die hier angegebenen Codes sind Zusatzinformationen, die für das ADS-Gerät spezifisch sind.
LEN	UDINT	Anzahl der zu schreibenden Daten in Bytes.
SRCADDR	PVOID	Adresse des Puffers, aus dem die zu schreibenden Daten geholt werden sollen. Der Programmierer ist selbst dafür verantwortlich, den Puffer in der Größe so zu dimensionieren, dass LEN Bytes daraus entnommen werden können. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
WRITE	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Zustandskennzahl des ADS-Zielgerätes

Die hier gezeigten Codes sind festgelegt für alle ADS-Server:

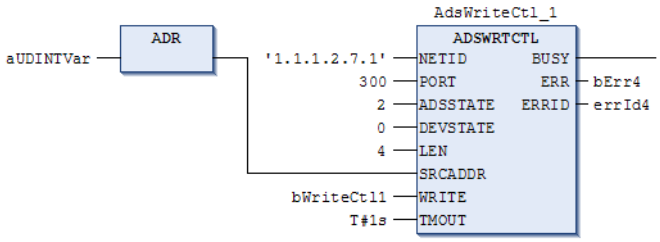
- ADSSTATE_IDLE = 1;
- ADSSTATE_RESET = 2;
- ADSSTATE_INIT = 3;
- ADSSTATE_START = 4;
- ADSSTATE_RUN= 5;
- ADSSTATE_STOP = 6;
- ADSSTATE_SAVECFG = 7;
- ADSSTATE_LOADCFG = 8;

Ausgänge

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

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE ist, wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein einen Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Beispiel für den Aufruf des Bausteins in FBD:



In dem Beispiel wird ein Reset-Kommando (ADSSTATE=2) an den IO-Server (Port 300) gesendet, mit den Zusatzdaten hex.AFFE. Der IO-Server führt daraufhin ein Bus-Reset aus.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.1.3 ADSRDDEVINFO

Der Funktionsbaustein liest die allgemeinen Geräteinformationen.

Eingänge

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

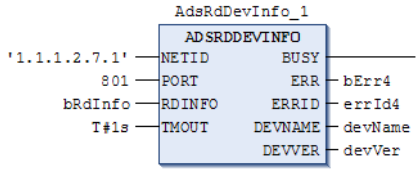
Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130]).
RDINFO	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

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

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.
DEV NAME	STRING	Name des ADS-Gerätes
DEVVER	UDINT	Versionsnummer des ADS-Gerätes

Beispiel für den Aufruf des Bausteins in FBD:



In dem Beispiel werden die Geräteinformationen des ersten SPS-Laufzeitsystems (Port 801) auf dem Rechner 1.1.1.2.7.1 gelesen. Als Ergebnis erhält man den Namen „PLC Server“ sowie die Versionsnummer 02.00.7 .

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.2 Indication + Response

3.2.2.1 Übersicht

Die erweiterten ADS-Funktionsbausteine ermöglichen den Aufbau einer Client-Server-Kommunikation zwischen einem ADS-Gerät und einer SPS-Task. Bei dem ADS-Gerät kann es sich z. B. um eine Windows-Applikation (nutzt die AdsDLL/AdsOcx) oder ein anderes SPS-Laufzeitsystem handeln. Die Kommunikation zwischen dem ADS-Gerät und der SPS-Task wird mittels folgenden Dienstprimitiven abgewickelt:

- Request
- Indication
- Response
- Confirmation

Die Kommunikation zwischen einem ADS-Gerät und einer SPS-Task hat folgenden Ablauf: Ein ADS-Gerät sendet ein Request (Anfrage) an das Zielgerät (SPS-Task). Diese Anfrage wird durch eine Indication in dem Zielgerät registriert. Das Zielgerät (SPS-Task) führt daraufhin einen entsprechenden Dienst aus. Der auszuführende Dienst wird über die Index-Group/Offset-Parameter verschlüsselt. Danach sendet die SPS-Task ein Response (Antwort) an das ADS-Gerät. Das Response wird von dem ADS-Quellgerät als Confirmation registriert.

Die ADS-Geräte werden über eine Portadresse und eine Netzwerkadresse (NETID) adressiert. (Portadresse der SPS-Task = _TaskInfo.AdsPort)

Damit ein Request an die SPS-Task weitergeleitet wird, muss in dem Index-Group-Parameter beim Request das höchstwertige Bit gesetzt werden (z.B. 0x80000001).

Kommunikation über IndexGroup 0x80000000 – 0x80FFFFFF

Pro SPS-Task kann sinnvoll nur eine Instanz des Indication- und Response-Funktionsbausteins benutzt werden (eine Instanz von ADSREADIND, ADSREADRES, ADSWRITEIND, ADSWRITERES, ADSRDWRTIND und ADSRDWRTRES). Entsprechend den verfügbaren ADS-Diensten: READ, WRITE und READ & WRITE gibt es zu jedem Dienst einen entsprechenden Indication- bzw. Response-Funktionsbaustein.

Dienst	Name	Beschreibung
READ	ADSREADIND [▶ 28]	ADSREAD-Indication
	ADSREADRES [▶ 37]	ADSREAD-Response
WRITE	ADSWRITEIND [▶ 31]	ADSWRITE-Indication
	ADSWRITERES [▶ 38]	ADSWRITE-Response
READ & WRITE	ADSRDWRTIND [▶ 34]	ADS-READ & WRITE-Indication
	ADSRDWRTRES [▶ 39]	ADS-READ & WRITE-Response

FiFos

Jede SPS-Task besitzt 3 Fifos in die die ankommenden Requests (Indications) zuerst abgelegt werden. Das heißt es gibt einen ADSREADIND-Fifo, ADSWRITEIND-Fifo und einen ADSRDWRTIND-Fifo.
In jedem Fifo können maximal 10 Indications gespeichert werden, bis diese abgearbeitet wurden (bis Response abgeschickt wurde). Wenn Sie z. B. gleichzeitig 12 ADSREAD-Requests an eine SPS-Task senden, dann werden 10 Requests als Indications in dem Fifo abgelegt und zwei mit der ADS-Fehlermeldung 1814 (0x716) quittiert (verworfen). In diesem Fall müssen Sie den Fehlercode auswerten und die zwei fehlgeschlagenen ADSREAD-Requests gegebenenfalls wiederholen. Durch den Aufruf der ADSxxxxxxIND-Instanz werden die Indications einzeln aus dem dazugehörigen Fifo rausgenommen. Erst danach können neue Indications erfolgreich in dem Fifo abgelegt werden.

Kommunikation über IndexGroup 0x8n000000 – 0x8nFFFFFF

Um mehr als eine Client-Server-Kommunikation pro SPS-Task zu realisieren, werden folgende Indication-Funktionsbausteine benötigt. Diese sind um die Möglichkeit erweitert, einen gewünschten Bereich der IndexGroup anzugeben.

So werden die Anfragen gefiltert und nur auf gewünschte Bereiche reagiert.

Es stehen hierbei 16 frei wählbare Bereiche zu Verfügung:
0x80000000 – 0x80FFFFFF
0x81000000 – 0x81FFFFFF

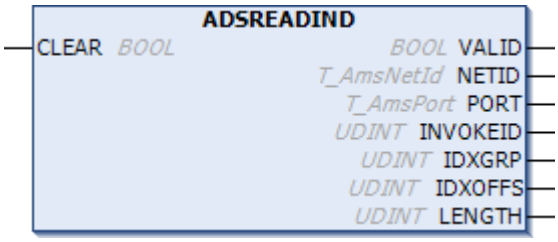
…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

Um an einem Indication-Funktionsbaustein einen solchen Bereich der Index-Group anzugeben, wird am Eingang MINIDXGRP der Index-Group-Wert angegeben, mit dem der gewählte Bereich beginnt.
Beispiel: Mit MINIDXGRP:=16#85000000 werden alle Anfragen gefiltert und Anfragen mit einer Index-Group im Bereich 0x85000000 – 0x85FFFFFF als Indication registriert.

Dienst	Name	Beschreibung
READ	ADSREADINDEX [▶ 29]	ADSREAD-Indication mit Angabe der Index-Group
	ADSREADRES [▶ 37]	ADSREAD-Response
WRITE	ADSWRITEINDEX [▶ 32]	ADSWRITE-Indication mit Angabe der Index-Group
	ADSWRITERES [▶ 38]	ADSWRITE-Response
READ & WRITE	ADSRDWRTINDEX [▶ 35]	ADS-READ & WRITE-Indication mit Angabe der Index-Group
	ADSRDWRTRES [▶ 39]	ADS-READ & WRITE-Response

3.2.2.2 ADSREADIND



Der Funktionsbaustein registriert ADSREAD-Anfragen (ADSREAD-Requests) an eine SPS-Task als Indications und erlaubt deren Bearbeitung. Das Anstehen einer Indication wird über eine steigende Flanke am VALID-Ausgang gemeldet. Über eine positive Flanke am CLEAR-Eingang wird die Indication als bearbeitet gemeldet. Eine negative Flanke am CLEAR-Eingang gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei. Nachdem eine Indication bearbeitet wurde, muss eine Antwort über den ADSREADRES [▶ 37]-Funktionsbaustein an das Quellgerät gesendet werden. Die Parameter PORT und NETID können dafür benutzt werden, um das Quellgerät zu adressieren. Der INVOKEID-Parameter dient dem Quellgerät der Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurückgesendet.

Eingänge

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

Name	Typ	Beschreibung
CLEAR	BOOL	Mit einer steigenden Flanke an diesem Eingang wird eine Indication als bearbeitet gemeldet und die Ausgänge des ADSREADIND-Funktionsbausteins zurückgesetzt. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei.

Ausgänge

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

Name	Typ	Beschreibung
VALID	BOOL	Der Ausgang ist gesetzt, wenn von dem Funktionsbaustein eine Indication registriert wurde und bleibt gesetzt, bis diese über eine positive Flanke an dem CLEAR-Eingang als bearbeitet gemeldet wurde.
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeID wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
LENGTH	UDINT	Anzahl der zu lesenden Daten in Bytes.

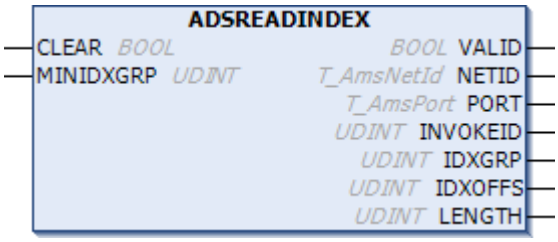
Beispiel für den Aufruf des Bausteins in ST:

- Beispiel mit AdsReadInd-/AdsReadRes-Funktionsbausteinen [▶ 141]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.2.3 ADSREADINDEX



Der Funktionsbaustein registriert ADSREAD-Anfragen (ADSREAD-Requests) an eine SPS-Task als Indications und erlaubt deren Bearbeitung. Das Anstehen einer Indication wird über eine steigende Flanke am VALID-Ausgang gemeldet. Über eine positive Flanke am CLEAR-Eingang wird die Indication als bearbeitet gemeldet. Eine negative Flanke am CLEAR-Eingang gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei. Nachdem eine Indication bearbeitet wurde, muss eine Antwort über den ADSREADRES [▶ 37]-Funktionsbaustein an das Quellgerät gesendet werden. Die Parameter PORT und NETID können dafür benutzt werden, um das Quellgerät zu adressieren. Der INVOKEID-Parameter dient dem Quellgerät der Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurückgesendet.

Gegenüber dem Funktionsbaustein ADSREADIND [▶ 28] besteht über einen zusätzlichen Eingang die Möglichkeit einen gewünschten Bereich der IndexGroup anzugeben.

So werden die Anfragen gefiltert und nur auf gewünschte Bereiche reagiert.

Es stehen hierbei 16 frei wählbare Bereiche zu Verfügung:
0x80000000 – 0x80FFFFFF
0x81000000 – 0x81FFFFFF

…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

Um an einem Indication-Funktionsbaustein einen solchen Bereich der Index-Group anzugeben, wird am Eingang MINIDXGRP der Index-Group-Wert angegeben, mit dem der gewählte Bereich beginnt.
Beispiel: Mit MINIDXGRP:=16#85000000 werden alle Anfragen gefiltert und Anfragen mit einer Index-Group im Bereich 0x85000000 – 0x85FFFFFF als Indication registriert.

Eingänge

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

Name	Typ	Beschreibung
CLEAR	BOOL	Mit einer steigenden Flanke an diesem Eingang wird eine Indication als bearbeitet gemeldet und die Ausgänge des ADSREADIND-Funktionsbausteins zurückgesetzt. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei.
MINIDXGRP	UDINT	Dieser Eingang ermöglicht die Filterung der Anfragen nach IndexGroup-Bereichen. Angabe des IndexGroup-Wertes, mit dem der gewählte Bereich beginnt.

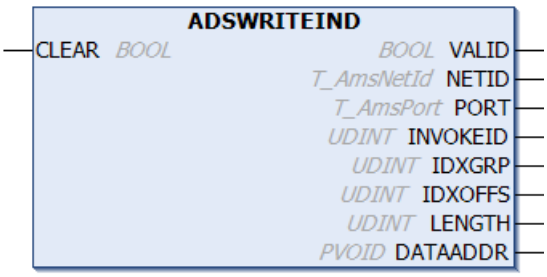
Ausgänge

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

Name	Typ	Beschreibung
VALID	BOOL	Der Ausgang ist gesetzt, wenn von dem Funktionsbaustein eine Indication registriert wurde und bleibt gesetzt, bis diese über eine positive Flanke an dem CLEAR-Eingang als bearbeitet gemeldet wurde.
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeID wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
LENGTH	UDINT	Anzahl der zu lesenden Daten in Bytes.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.35	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.26.0

3.2.2.4 ADSWRITEIND



Der Funktionsbaustein registriert ADSWRITE-Anfragen (ADSWRITE-Requests) an eine SPS-Task als Indications und erlaubt deren Bearbeitung. Das Anstehen einer Indication wird über eine steigende Flanke am VALID-Ausgang gemeldet. Über eine positive Flanke am CLEAR-Eingang wird die Indication als bearbeitet gemeldet. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei. Nachdem eine Indication bearbeitet wurde, muss eine Antwort über den ADSWRITERES [▶ 38]-Funktionsbaustien an das Quellgerät gesendet werden. Die Parameter: PORT, NETID können dafür benutzt werden, um das Quellgerät zu adressieren. Der INVOKEID-Parameter dient dem Quellgerät der Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurück gesendet.

Eingänge

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

Name	Typ	Beschreibung
CLEAR	BOOL	Mit einer steigenden Flanke an diesem Eingang wird eine Indication als bearbeitet gemeldet und die Ausgänge des ADSWRITEIND-Funktionsbausteins zurückgesetzt (DATAADDR = 0, LENGTH = 0 !). Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei.

Ausgänge

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

Name	Typ	Beschreibung
VALID	BOOL	Der Ausgang ist gesetzt, wenn von dem Funktionsbaustein eine Indication registriert wurde und bleibt gesetzt, bis diese über eine positive Flanke an dem CLEAR-Eingang als bearbeitet gemeldet wurde.
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeID wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
LENGTH	UDINT	Länge der geschriebenen Daten in Bytes.
DATAADDR	PVOID	Adresse des Datenpuffers, in dem sich die geschriebenen Daten befinden.

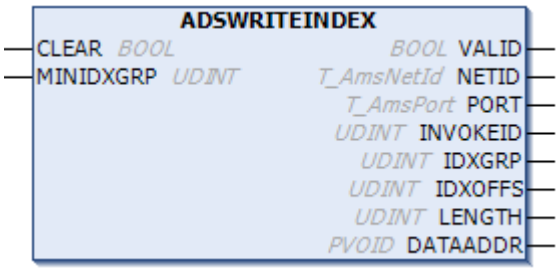
Beispiel für den Aufruf des Bausteins in ST:

- Beispiel mit AdsWriteInd-/AdsWriteRes-Funktionsbausteinen [▶ 143]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.2.5 ADSWRITEINDEX



Der Funktionsbaustein registriert ADSWRITE-Anfragen (ADSWRITE-Requests) an eine SPS-Task als Indications und erlaubt deren Bearbeitung. Das Anstehen einer Indication wird über eine steigende Flanke am VALID-Ausgang gemeldet. Über eine positive Flanke am CLEAR-Eingang wird die Indication als bearbeitet gemeldet. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei. Nachdem eine Indication bearbeitet wurde, muss eine Antwort über den ADSWRITERES [▶ 38]-Funktionsbaustien an das Quellgerät gesendet werden. Die Parameter: PORT, NETID können dafür benutzt werden, um das Quellgerät zu adressieren. Der INVOKEID-Parameter dient dem Quellgerät der Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurück gesendet.

Gegenüber dem Funktionsbaustein ADSWRITEIND [▶ 31] besteht über einen zusätzlichen Eingang die Möglichkeit einen gewünschten Bereich der IndexGroup anzugeben.

So werden die Anfragen gefiltert und nur auf gewünschte Bereiche reagiert.

Es stehen hierbei 16 frei wählbare Bereiche zu Verfügung:
0x80000000 – 0x80FFFFFF
0x81000000 – 0x81FFFFFF

…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

Um an einem Indication-Funktionsbaustein einen solchen Bereich der Index-Group anzugeben, wird am Eingang MINIDXGRP der Index-Group-Wert angegeben, mit dem der gewählte Bereich beginnt.
Beispiel: Mit MINIDXGRP:=16#85000000 werden alle Anfragen gefiltert und Anfragen mit einer Index-Group im Bereich 0x85000000 – 0x85FFFFFF als Indication registriert.

Eingänge

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

Name	Typ	Beschreibung
CLEAR	BOOL	Mit einer steigenden Flanke an diesem Eingang wird eine Indication als bearbeitet gemeldet und die Ausgänge des ADSWRITEIND-Funktionsbausteins zurückgesetzt (DATAADDR = 0, LENGTH = 0 !). Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei.
MINIDXGRP	UDINT	Dieser Eingang ermöglicht die Filterung der Anfragen nach IndexGroup-Bereichen. Angabe des IndexGroup-Wertes, mit dem der gewählte Bereich beginnt.

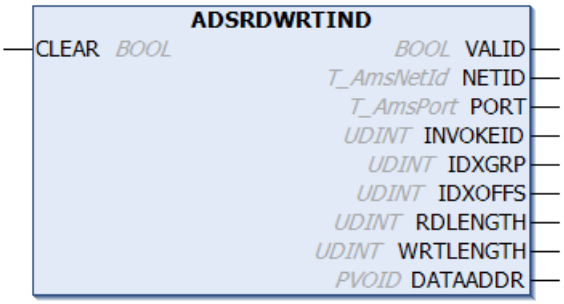
Ausgänge

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

Name	Typ	Beschreibung
VALID	BOOL	Der Ausgang ist gesetzt, wenn von dem Funktionsbaustein eine Indication registriert wurde und bleibt gesetzt, bis diese über eine positive Flanke an dem CLEAR-Eingang als bearbeitet gemeldet wurde.
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeID wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
LENGTH	UDINT	Länge der geschriebenen Daten in Bytes.
DATAADDR	PVOID	Adresse des Datenpuffers, in dem sich die geschriebenen Daten befinden.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.35	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.26.0

3.2.2.6 ADSRDWRTIND



Der Funktionsbaustein registriert ADSRDWRT-Anfragen (ADSRDWRT-Requests) an eine SPS-Task als Indications und erlaubt deren Bearbeitung. Das Anstehen einer Indication wird über eine steigende Flanke am VALID-Ausgang gemeldet. Über eine positive Flanke am CLEAR-Eingang wird die Indication als bearbeitet gemeldet. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei. Nachdem eine Indication bearbeitet wurde, muss eine Antwort über den ADSRDWRTRES [▶ 39]-Funktionsbaustein an das Quellgerät gesendet werden. Die Parameter PORT, NETID können dafür benutzt werden, um das Quellgerät zu adressieren. Der INVOKEID-Parameter dient dem Quellgerät zur Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurück gesendet.

Eingänge

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

Name	Typ	Beschreibung
CLEAR	BOOL	Mit einer steigenden Flanke an diesem Eingang wird eine Indication als bearbeitet gemeldet und die Ausgänge des ADSRDWRTIND-Funktionsbausteins zurückgesetzt. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei.

Ausgänge

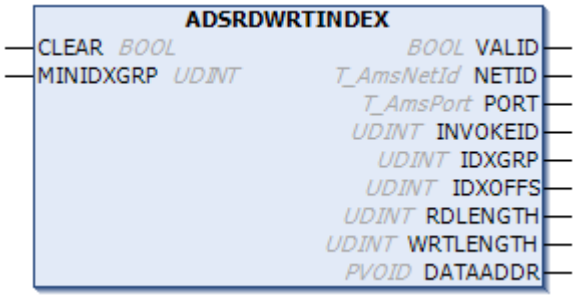
```
VAR_OUTPUT
    VALID     : BOOL;
    NETID     : T_AmsNetId;
    PORT      : T_AmsPort;
    INVOKEID  : UDINT;
    IDXGRP    : UDINT;
    IDXOFFS   : UDINT;
    RDLENGTH  : UDINT;
    WRTLENGTH : UDINT;
    DATAADDR  : PVOID;
END_VAR
```

Name	Typ	Beschreibung
VALID	BOOL	Der Ausgang ist gesetzt, wenn von dem Funktionsbaustein eine Indication registriert wurde und bleibt gesetzt, bis diese über eine positive Flanke an dem CLEAR-Eingang als bearbeitet gemeldet wurde.
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
RDLENGTH	UDINT	Länge der zu lesenden Daten in Bytes.
WRTLENGTH	UDINT	Länge der geschriebenen Daten in Bytes.
DATAADDR	PVOID	Adresse des Datenpuffers, in dem sich die geschriebenen Daten befinden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.2.7 ADSRDWRTINDEX



Der Funktionsbaustein registriert ADSRDWRT-Anfragen (ADSRDWRT-Requests) an eine SPS-Task als Indications und erlaubt deren Bearbeitung. Das Anstehen einer Indication wird über eine steigende Flanke am VALID-Ausgang gemeldet. Über eine positive Flanke am CLEAR-Eingang wird die Indication als bearbeitet gemeldet. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei. Nachdem eine Indication bearbeitet wurde, muss eine Antwort über den ADSRDWRTRES [▶ 39]-Funktionsbaustein an das Quellgerät gesendet werden. Die Parameter PORT, NETID können dafür benutzt werden, um das Quellgerät zu adressieren. Der INVOKEID-Parameter dient dem Quellgerät zur Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurück gesendet.

Gegenüber dem Funktionsbaustein ADSRDWRTIND [▶ 34] besteht über einen zusätzlichen Eingang die Möglichkeit einen gewünschten Bereich der IndexGroup anzugeben.

So werden die Anfragen gefiltert und nur auf gewünschte Bereiche reagiert.

Es stehen hierbei 16 frei wählbare Bereiche zu Verfügung:
0x80000000 – 0x80FFFFFF
0x81000000 – 0x81FFFFFF

…

0x8E000000 – 0x8EFFFFFF
0x8F000000 – 0x8FFFFFFF

Um an einem Indication-Funktionsbaustein einen solchen Bereich der Index-Group anzugeben, wird am Eingang MINIDXGRP der Index-Group-Wert angegeben, mit dem der gewählte Bereich beginnt.
Beispiel: Mit MINIDXGRP:=16#85000000 werden alle Anfragen gefiltert und Anfragen mit einer Index-Group im Bereich 0x85000000 – 0x85FFFFFF als Indication registriert.

Eingänge

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

Name	Typ	Beschreibung
CLEAR	BOOL	Mit einer steigenden Flanke an diesem Eingang wird eine Indication als bearbeitet gemeldet und die Ausgänge des ADSRDWRTIND-Funktionsbausteins zurückgesetzt. Eine fallende Flanke gibt den Funktionsbaustein für die Verarbeitung weiterer Indications frei.
MINIDXGRP	UDINT	Dieser Eingang ermöglicht die Filterung der Anfragen nach IndexGroup-Bereichen. Angabe des IndexGroup-Werts, mit dem der gewählte Bereich beginnt.

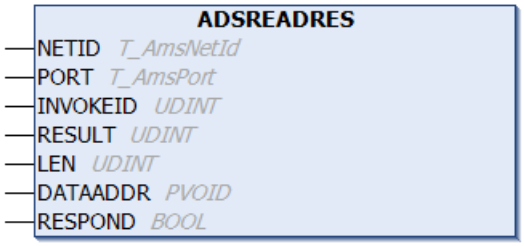
Ausgänge

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

Name	Typ	Beschreibung
VALID	BOOL	Der Ausgang ist gesetzt, wenn von dem Funktionsbaustein eine Indication registriert wurde und bleibt gesetzt, bis diese über eine positive Flanke an dem CLEAR-Eingang als bearbeitet gemeldet wurde.
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, von dem der ADS-Befehl gesendet wurde (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes.
RDLENGTH	UDINT	Länge der zu lesenden Daten in Bytes.
WRTLENGTH	UDINT	Länge der geschriebenen Daten in Bytes.
DATAADDR	PVOID	Adresse des Datenpuffers, in dem sich die geschriebenen Daten befinden.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.35	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.26.0

3.2.2.8 ADSREADRES



Der Funktionsbaustein ADSREADRES quittiert Indications einer SPS-Task. Über eine positive Flanke am RESPOND-Eingang wird eine Antwort an das ADS-Quellgerät gesendet. Das Quellgerät wird über die Parameter: PORT und NETID adressiert. Der Parameter INVOKEID dient dem Quellgerät zur Zuordnung der Antworten zu den Anfragen und wird von dem Ausgang des ADSREADIND [▶ 28]-Funktionsbaustein übernommen. Über den RESULT-Parameter kann ein Fehlercode an das ADS-Quellgerät zurückgegeben werden.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, an den der ADS-Befehl gesendet werden soll (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, an den die Antwort gesendet werden soll (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeID wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
RESULT	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode, der an das Quellgerät gesendet werden soll.
LEN	UDINT	Anzahl der zu lesenden Daten in Bytes.
DATAADDR	PVOID	Adresse des Datenpuffers, der gelesen werden sollte.
RESPOND	BOOL	Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

Ausgänge

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

Beispiel für den Aufruf des Bausteins in ST:

- Beispiel mit AdsReadInd-/AdsReadRes-Funktionsbausteinen [▶ 141]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.2.9 ADSWRITERES

Der Funktionsbaustein ADSWRITERES quittiert Indications einer SPS-Task. Über eine positive Flanke am RESPOND-Eingang wird eine Antwort an das ADS-Quellgerät gesendet. Das Quellgerät wird über die Parameter: PORT und NETID adressiert. Der Parameter INVOKEID dient dem Quellgerät der Zuordnung der Antworten zu den Anfragen und wird von dem Ausgang des ADSWRITEIND [▶ 28]-Funktionsbausteins übernommen. Über den RESULT-Parameter kann ein Fehlercode an das ADS-Quellgerät zurückgegeben werden.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, an den der ADS-Befehl gesendet werden soll (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, an den der ADS-Befehl gesendet werden soll (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
RESULT	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode, der an das Quellgerät gesendet werden soll.
RESPOND	BOOL	Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

Ausgänge

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

Beispiel für den Aufruf des Bausteins in ST:

- Beispiel mit AdsWriteInd-/AdsWriteRes-Funktionsbausteinen [▶ 143]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.2.10 ADSRDWRTRES

Der Funktionsbaustein ADSRDWRTRES quittiert Indications einer SPS-Task. Über eine positive Flanke am RESPOND-Eingang wird eine Antwort an das ADS-Quellgerät gesendet. Das Quellgerät wird über die Parameter: PORT und NETID adressiert. Der Parameter INVOKEID dient dem Quellgerät der Zuordnung der Antworten zu den Anfragen und wird von dem Ausgang des ADSRDWRTIND [▶ 34]-Funktionsbausteins übernommen. Über den RESULT-Parameter kann ein Fehlercode an das ADS-Quellgerät zurückgegeben werden.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Quellgerätes enthält, an den der ADS-Befehl gesendet werden soll (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Quellgerätes, an den der ADS-Befehl gesendet werden soll (Typ: T_AmsPort [▶ 130]).
INVOKEID	UDINT	Handle des Befehls, der gesendet wurde. Die InvokeID wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.
RESULT	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode, der an das Quellgerät gesendet werden soll.
LEN	UDINT	Länge der gelesenen Daten in Byte.
DATAADDR	PVOID	Adresse des Datenpuffers, in dem sich die gelesenen Daten befinden.
RESPOND	BOOL	Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

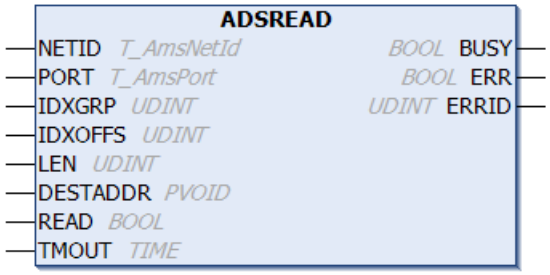
Ausgänge

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.3 ADSREAD



Der Funktionsbaustein führt einen ADS-Lesebefehl aus, um Daten von einem ADS-Gerät anzufordern.

Veraltete Response-Daten

Nach einer Unterbrechung der Verbindung werden bei erneuter Verbindung die alten Response-Daten ausgegeben.
Um das zu verhindern, achten Sie darauf, nicht dieselbe ADS-Read Instanz für mehrere Targets zu verwenden.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130])
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
LEN	UDINT	Anzahl der zu lesenden Daten in Bytes
DESTADDR	PVOID	Adresse des Puffers, der die gelesenen Daten aufnehmen soll. Der Programmierer ist selbst dafür verantwortlich den Puffer in der Größe so zu dimensionieren, dass er LEN Bytes aufnehmen kann. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
READ	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

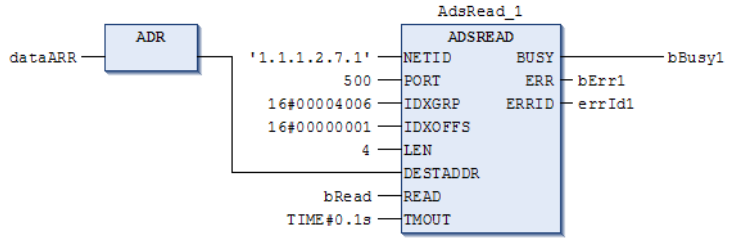
```
VAR_OUTPUT
    BUSY     : BOOL;
    ERR      : BOOL;
    ERRID    : UDINT;
END_VAR
```

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE ist, wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein einen Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Beispiel für den Aufruf des Bausteins in ST:

- Beispiel mit AdsRead-Funktionsbaustein [▶ 144]

Beispiel für den Aufruf des Bausteins in FBD:

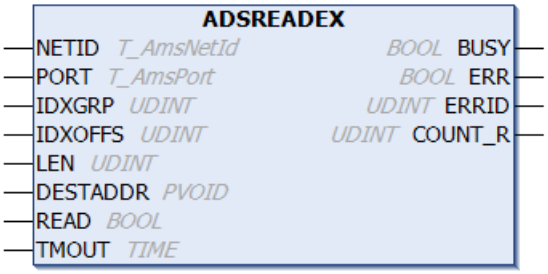


Hierbei wird der Fehlerstatus der Achse Nr. 6 als 4 Byte großes Element abgefragt und in den Puffer dataArr geschrieben. Der IDXGRP 00004006(hex) und der IDXOFFS 00000001(hex) ergeben sich aus der NC-ADS-Dokumentation.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.4 ADSREADEX



Der Funktionsbaustein führt einen ADS-Lesebefehl aus, um Daten von einem ADS-Gerät anzufordern. Der Funktionsbaustein hat die gleiche Funktionalität wie der ADSREAD-Funktionsbaustein, liefert aber zusätzlich auch die Anzahl der tatsächlich gelesenen Datenbytes als Parameter zurück.

Eingänge

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

```
    IDXOFFS  : UDINT;
    LEN      : UDINT;
    DESTADDR : PVOID;
    READ     : BOOL;
    TMOUT    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130])
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
LEN	UDINT	Anzahl der zu lesenden Daten in Bytes
DESTADDR	PVOID	Adresse des Puffers, der die gelesenen Daten aufnehmen soll. Der Programmierer ist selbst dafür verantwortlich den Puffer in der Größe so zu dimensionieren, dass er LEN Bytes aufnehmen kann. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
READ	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

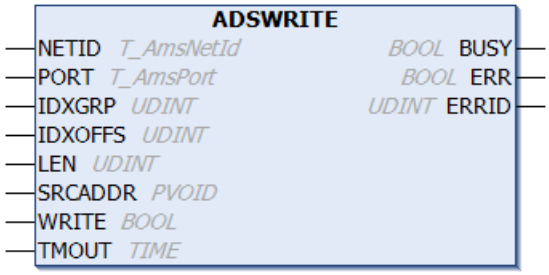
```
VAR_OUTPUT
    BUSY    : BOOL;
    ERR     : BOOL;
    ERRID   : UDINT;
    COUNT_R : UDINT;
END_VAR
```

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.
COUNT_R	UDINT	Anzahl der erfolgreich gelesenen Datenbytes

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.5 ADSWRITE



Baustein zur Ausführung eines ADS-Schreibbefehls, um Daten zu einem ADS-Gerät zu übermitteln.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130])
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
LEN	UDINT	Anzahl der zu lesenden Daten in Bytes
SRCADDR	PVOID	Adresse des Puffers, aus dem die zu schreibenden Daten geholt werden sollen. Der Programmierer ist selbst dafür verantwortlich, den Puffer in der Größe so zu dimensionieren, dass LEN Bytes daraus entnommen werden können. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
WRITE	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

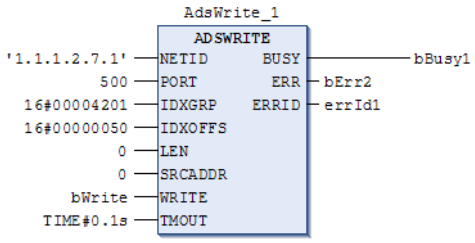
```
VAR_OUTPUT
    BUSY    : BOOL;
    ERR     : BOOL;
    ERRID   : UDINT;
END_VAR
```

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE ist, wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein einen Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Beispiel für den Aufruf des Bausteins in ST:

- Beispiel mit AdsWrite-Funktionsbaustein [▶ 145]

Beispiel für den Aufruf des Bausteins in FBD:

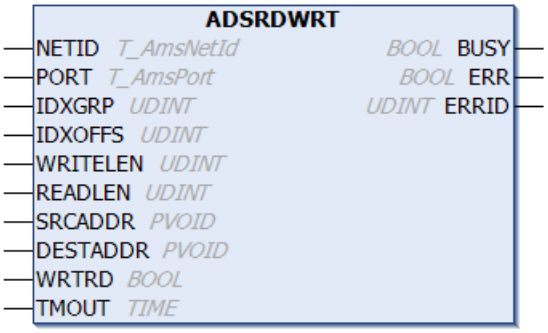


Hierbei wird die NC-Achse Nr. 1 durch einen Schreibbefehl mit IDXGRP 00004201(hex) und der IDXOFFS 00000050(hex) deaktiviert. Zur Aktivierung der Achse müsste ein erneuter Schreibbefehl mit dem IDXOFFS 00000051(hex)) erfolgen. Da dieser Schreibbefehl keine Parameter benötigt, sind die Eingänge LEN und SRCADDR ohne Bedeutung, müssen aber auf Null gesetzt werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.6 ADSRDWRT



Der Funktionsbaustein führt einen kombinierten ADS-Schreib-Lesebefehl aus. Mit einem Aufruf werden Daten zu einem ADS-Gerät übermittelt (Write) und dessen Antwortdaten gelesen (Read).

Eingänge

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

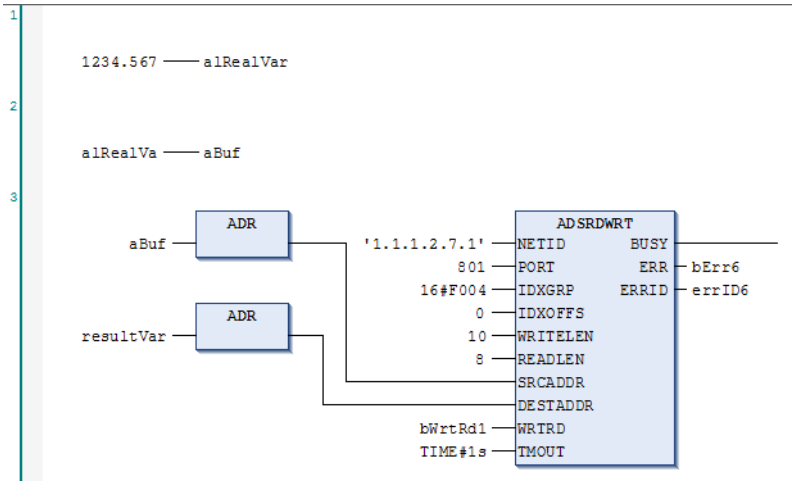
Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130])
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
WRITELEN	UDINT	Anzahl der zu schreibenden Daten in Bytes
READLEN	UDINT	Anzahl der zu lesenden Daten in Bytes
SRCADDR	PVOID	Adresse des Puffers, aus dem die zu schreibenden Daten geholt werden sollen. Der Programmierer ist selbst dafür verantwortlich, den Puffer in der Größe so zu dimensionieren, dass WRITELEN Bytes daraus entnommen werden können. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
DESTADDR	PVOID	Adresse des Puffers, der die gelesenen Daten aufnehmen soll. Der Programmierer ist selbst dafür verantwortlich den Puffer in der Größe so zu dimensionieren, dass er READLEN Bytes aufnehmen kann. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
WRTRD	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

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

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE ist, wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein einen Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Beispiel für den Aufruf des Bausteins in FBD:

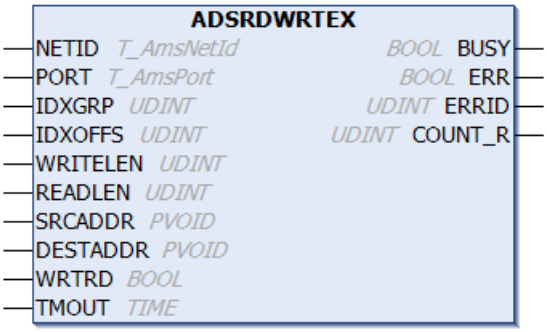


Hierbei soll der Wert der Variablen mit dem Namen „aLRealVar“ aus der SPS, die auf dem Rechner mit der Net-Id 1.1.1.2.7.1 läuft, gelesen werden. Dazu wird die erwähnte Rechneradresse sowie die Portnummer des ersten Laufzeitsystems der SPS, der Index-Group, Index-Offset für namentliches Variablen lesen (F004 hex,0) angegeben. Der Name der Variablen soll zum PLC-Server übertragen werden; dazu wird er in einen Puffer abgelegt. Da die Variable global ist, erhält sie einen führenden Punkt. Die Länge der zu schreibenden Daten sind somit 10 (1 Punkt sowie 9 Buchstaben). Da die zu lesende Variable ein LREAL-Typ ist, beträgt die Anzahl der zu lesenden Bytes 8. Als Adresse für die zu schreibenden Daten wird die Adresse des Namenspuffers angegeben, als Adresse für die Empfangsdaten wird die Adresse einer LREAL-Variablen (resultVar) angegeben. Die Abbildung zeigt den Zustand des Bausteins in Ablaufkontrolle nach Ausführung des WriteRead-Befehls: Der Wert 1234.567, der vorher in aLRealVar enthalten war, ist jetzt auch in resultVar enthalten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.2.7 ADSRDWRTEX



Der Baustein erlaubt die Ausführung eines kombinierten ADS-Schreib-Lesebefehls. Mit einem Aufruf werden Daten zu einem ADS-Gerät übermittelt (Write) und dessen Antwortdaten gelesen (Read). Im Gegensatz zu dem ADSRDWRT-Funktionsbaustein liefert ADSRDWRTEX die Anzahl der tatsächlich gelesenen Datenbytes als Parameter zurück.

Eingänge

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

```
    WRITELEN : UDINT;
    READLEN  : UDINT;
    SRCADDR  : PVOID;
    DESTADDR : PVOID;
    WRTRD    : BOOL;
    TMOUT    : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
```

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes (Typ: T_AmsPort [▶ 130])
IDXGRP	UDINT	Index-Gruppennummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
IDXOFFS	UDINT	Index-Offsetnummer (32 Bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.
WRITELEN	UDINT	Anzahl der zu schreibenden Daten in Bytes
READLEN	UDINT	Anzahl der zu lesenden Daten in Bytes
SRCADDR	PVOID	Adresse des Puffers, aus dem die zu schreibenden Daten geholt werden sollen. Der Programmierer ist selbst dafür verantwortlich, den Puffer in der Größe so zu dimensionieren, dass WRITELEN Bytes daraus entnommen werden können. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
DESTADDR	PVOID	Adresse des Puffers, der die gelesenen Daten aufnehmen soll. Der Programmierer ist selbst dafür verantwortlich den Puffer in der Größe so zu dimensionieren, dass er READLEN Bytes aufnehmen kann. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
WRTRD	BOOL	Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

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

Name	Typ	Beschreibung
BUSY	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der an dem Timeout-Eingang angelegten Zeit. Während BUSY = TRUE wird an den Eingängen kein neuer Befehl angenommen. Beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.
ERR	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ERRID enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist ERR = TRUE und ERRID = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ERRID	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.
COUNT_R	UDINT	Anzahl der erfolgreich gelesenen Datenbytes

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3 Datei-Funktionsbausteine

Diese Funktionsbausteine sind nur bedingt für ein Logging in Echtzeit geeignet.
Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

3.3.1 FB_EOF

Der Funktionsbaustein FB_EOF kann überprüfen, ob das Ende der Datei erreicht wurde.

Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

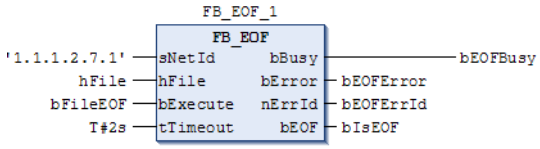
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
    bEOF   : BOOL;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
bEOF	BOOL	Dieser Ausgang wird gesetzt, wenn das Dateiende erreicht worden ist.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in FBD:

```
PROGRAM Test
VAR
    fbEOF        : FB_EOF;
    hFile        : UINT;
    bFileEOF     : BOOL;
    bEOFBusy     : BOOL;
    bEOFError    : BOOL;
    nEOFErrorId  : UDINT;
    bIsEOF       : BOOL;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.2 FB_FileOpen

Der Funktionsbaustein FB_FileOpen legt eine neue Datei an bzw. öffnet eine bereits bestehende Datei zur weiteren Bearbeitung.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
sPathName	T_MaxString	Speicherpfad und Dateiname der zu öffnenden Datei. Der Pfad kann nur auf das lokale Dateisystem des Rechners zeigen. Netzwerkpfade können hier nicht angegeben werden (Typ: T_MaxString [▶ 132]).
nMode	DWORD	Modus für das Öffnen der Datei.
ePath	E_OpenPath	Über diesen Eingang kann ein TwinCAT-Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden (Typ: E_OpenPath [▶ 126]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Vordefinierte Öffnungsmodi für nMode

Modus für das Öffnen der Datei. Die nachfolgend aufgeführten Codes sind die verschiedenen Öffnungsmodi, die bereits in der Bibliothek als Konstanten vordefiniert sind, und dem Baustein dementsprechend symbolisch übergeben werden können. Die Öffnungsmodi können durch ODER-Verknüpfung kombiniert werden. Die Öffnungsmodi können auf ähnliche Weise wie die Öffnungsmodi der fopen-Funktion in C bzw. C++ kombiniert werden.

Modi	Beschreibung
FOPEN_MODEREAD	r: Öffnet eine Datei zum Lesen. Wenn die Datei nicht gefunden werden kann oder nicht existiert, wird ein Fehler zurückgeliefert.
FOPEN_MODEWRITE	w: Öffnet eine leere Datei zum Schreiben. Wenn die Datei bereits existiert, dann wird sie überschrieben.
FOPEN_MODEAPPEND	a: Öffnet eine Datei zum Schreiben am Ende der Datei (anhängen). Wenn die Datei nicht existiert, wird zuerst eine neue angelegt.
FOPEN_MODEREAD OR FOPEN_MODEPLUS	r+: Öffnet eine Datei zum Lesen und zum Schreiben. Die Datei muss existieren.
FOPEN_MODEWRITE OR FOPEN_MODEPLUS	w+: Öffnet eine leere Datei zum Lesen und zum Schreiben. Wenn die Datei bereits existiert, dann wird sie überschrieben.
FOPEN_MODEAPPEND OR FOPEN_MODEPLUS	a+: Öffnet eine Datei zum Lesen und zum Schreiben am Ende der Datei (anhängen). Wenn die Datei nicht existiert, wird zuerst eine neue angelegt. Dazu muss der Speicherpfad bekannt sein, ansonsten erscheint der Fehler 1804. Alle Schreiboperationen werden immer am Ende einer Datei ausgeführt, wenn diese Datei in dem Modi a oder a+ geöffnet wurde. Der Dateizeiger kann mit FB_FileSeek versetzt werden, er wird aber beim Schreiben immer zuerst ans Ende der Datei bewegt. D. h. existierende Daten können nicht überschrieben werden.
FOPEN_MODEBINARY	b: Öffnet die Datei im Binär-Mode
FOPEN_MODETEXT	t: Öffnet die Datei im Text-Mode

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
hFile	UINT	Enthält, nach erfolgreichem Öffnen, das erzeugte File-Handle für die Datei. Dieses Handle wird dann an die anderen File-Funktionsbausteine als Kennung für die zu bearbeitete Datei übergeben.

Fehlercodes für hFile

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Unbekannter oder ungültiger nMode oder ePath Parameter.
0x70C	Datei nicht gefunden. Ungültiger Dateiname oder Dateipfad.
0x716	Keine weiteren freien File-Handles.

Information:

Bei dem Öffnungsmodus dürfen maximal 3 Parameter mit ODER verknüpft werden:

Mode = [Parameter1] OR [Parameter2] OR [Paramerter3]
Parameter1 darf nur einen unterstehenden Wert haben:

- FOPEN_MODEREAD
- FOPEN_MODEWRITE
- FOPEN_MODEAPPEND

Parameter2 darf nur einen unterstehenden Wert haben:

- FOPEN_MODEPLUS

Parameter3 darf nur einen unterstehenden Wert haben:

- FOPEN_MODEBINARY
- FOPEN_MODETEXT

Wenn kein Binär- oder Text-Mode angegeben wurde, öffnet die Datei in einem Mode der durch eine Betriebssystemvariable festgelegt ist. In den meisten Fällen öffnet die Datei dann im Text-Mode. Eine eindeutige Aussage ist jedoch nicht möglich. Es ist sinnvoll den Text oder Binärmode immer anzugeben. Diese Systemvariable kann nicht in der SPS geändert werden!
Daraus ergeben sich folgende zulässige Kombinationen:

Textdatei Öffnungsmodi	Binärdatei Öffnungsmodi
FOPEN_MODEREAD OR FOPEN_MODETEXT	FOPEN_MODEREAD OR FOPEN_MODEBINARY
FOPEN_MODEWRITE OR FOPEN_MODETEXT	FOPEN_MODEWRITE OR FOPEN_MODEBINARY
FOPEN_MODEAPPEND OR FOPEN_MODETEXT	FOPEN_MODEAPPEND OR FOPEN_MODEBINARY
FOPEN_MODEREAD OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEREAD OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY
FOPEN_MODEWRITE OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEWRITE OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY
FOPEN_MODEAPPEND OR FOPEN_MODEPLUS OR FOPEN_MODETEXT	FOPEN_MODEAPPEND OR FOPEN_MODEPLUS OR FOPEN_MODEBINARY

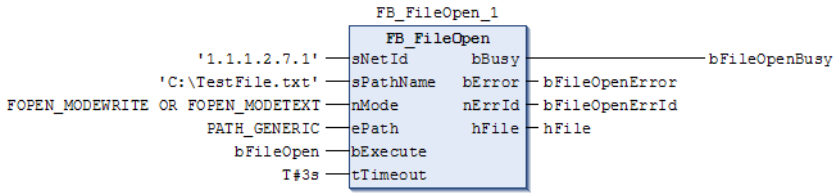
Alle anderen Kombinationen sind falsch. Beispiele für unzulässige Öffnungsmodi:
FOPEN_MODEBINARY OR FOPEN_MODETEXT
FOPEN_MODEWRITE OR FOPEN_MODEAPPEND

Beispiel für den Aufruf des Bausteins in ST:

- Dateizugriff aus der SPS [▶ 147]

Beispiel für den Aufruf des Bausteins in FBD:

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



Hierbei soll die Datei *TestFile2.txt* im Root-Verzeichnis des Laufwerks C: im Textmode angelegt bzw. überschrieben werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.3 FB_FileClose

Der Funktionsbaustein FB_FileClose schließt die Datei und versetzt sie damit in einen definierten Zustand zur weiteren Verarbeitung durch andere Programme.

Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Handle der Datei, die geschlossen werden soll.
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischen Fehlercode für nErrId

Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Möglicher Ursache
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

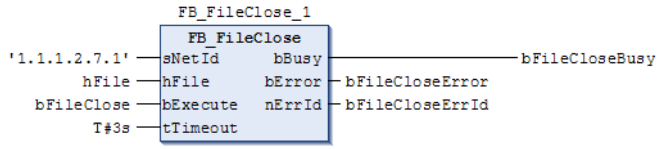
Beispiel für den Aufruf des Bausteins in ST:

- Dateizugriff aus der SPS [▶ 147]

Beispiel für den Aufruf des Bausteins in FBD:

```
PROGRAM Test
VAR
    fbFileClose        : FB_FileClose;
    hFile              : UINT;
```

```
    bFileClose         : BOOL;
    bFileCloseBusy     : BOOL;
    bFileCloseError    : BOOL;
    nFileCloseErrorId  : UDINT;
END_VAR
```



Hier wird die Datei, welche mit dem (durch FB_FileOpen) File-Handle verknüpft ist, wieder geschlossen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.4 FB_FileLoad

Mit dem Funktionsbaustein FB_FileLoad kann der Inhalt einer Datei ausgelesen werden. Die Datei wird implizit im Binär-Mode geöffnet, der Inhalt ausgelesen und die Datei daraufhin geschlossen.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
sPathName	T_MaxString	Speicherpfad und Dateiname der zu öffnenden Datei. Der Pfad kann nur auf das lokale Dateisystem des Rechners zeigen. Netzwerkpfade können hier nicht angegeben werden (Typ: T_MaxString [▶ 132])
pReadBuff	PVOID	Adresse des Puffers, in den die Daten gelesen werden sollen. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
cbReadLen	UDINT	Anzahl der zu lesenden Bytes.
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung der internen ADS-Kommandos nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
cbRead	UDINT	Anzahl der aktuell gelesenen Bytes

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4022.0	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= v3.4.22.0

3.3.5 FB_FileGets

Der Funktionsbaustein FB_FileGest liest Strings aus einer Datei. Der String wird bis zum Zeilenvorschub-Character und inklusive des Zeilenvorschub-Characters oder bis zum Ende der Datei oder bis zur maximal zulässigen Länge von sLine gelesen. Die Null-Terminierung wird automatisch angehängt. Die Datei muss dafür im Textmodus geöffnet worden sein.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

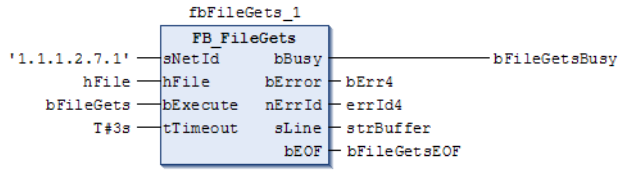
```
VAR_OUTPUT
    bBusy    : BOOL;
    bError   : BOOL;
    nErrId   : UDINT;
    sLine    : T_MaxString;
    bEOF     : BOOL;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn ein Fehler bei der Ausführung des Befehls auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
sLine	T_MaxString	String, der gelesen wurde (Typ: T_MaxString [▶ 132]).
bEOF	BOOL	Dieser Ausgang wird gesetzt, wenn das Dateiende erreicht worden ist und keine weiteren Datenbytes gelesen werden konnten (cbRead=0). Dieser Ausgang wird nicht gesetzt, wenn noch Datenbytes gelesen werden konnten (cbRead>0).

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70A	No memory for read buffer.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in FBD:

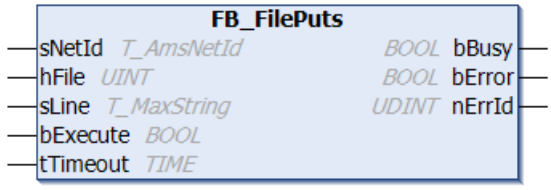
```
PROGRAM Test
VAR
    fbFileGets          : FB_FileGets;
    hFile               : UINT;
    bFileGets           : BOOL;
    bFileGetsBusy       : BOOL;
    bFileGetsError      : BOOL;
    nFileGetsErrorId    : UDINT;
    strBuffer           : STRING;
    bFileGetsEOF        : BOOL;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.6 FB_FilePuts



Der Funktionsbaustein FB_FilePuts schreibt Strings in eine Datei. Der String wird bis zur Null-Terminierung aber ohne den Null-Character in die Datei geschrieben. Die Datei muss dafür im Textmodus geöffnet worden sein.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
sLine	T_MaxString	String, der in die Datei geschrieben werden soll (Typ: T_MaxString [▶ 132]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

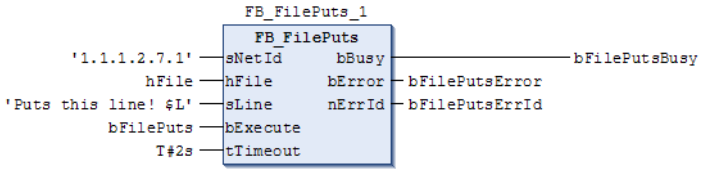
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in FBD:

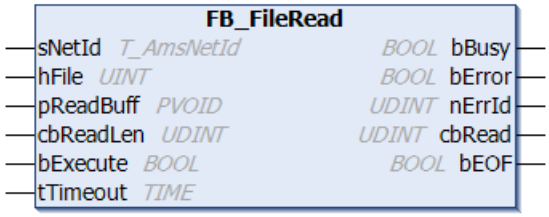
```
PROGRAM Test
VAR
    fbFilePuts        : FB_FilePuts;
    hFile             : UINT;
    bFilePuts         : BOOL;
    bFilePutsBusy     : BOOL;
    bFilePutsError    : BOOL;
    nFilePutsErrorId  : UDINT;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.7 FB_FileRead



Mit dem Funktionsbaustein FB_FileRead kann der Inhalt einer bereits geöffneten Datei ausgelesen werden. Die Datei muss vor einem Lesezugriff im entsprechenden Modus geöffnet worden sein. Dabei ist neben dem `FOPEN_MODEREAD` auch das passende Format (`FOPEN_MODEBINARY` oder `FOPEN_MODETEXT`) wichtig, um das gewünschte Resultat zu erzielen.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
pReadBuff	PVOID	Adresse des Puffers, in den die Daten gelesen werden sollen. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
cbReadLen	UDINT	Anzahl der zu lesenden Bytes
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
cbRead	UDINT	Anzahl der aktuell gelesenen Bytes
bEOF	BOOL	Dieser Ausgang wird gesetzt, wenn das Dateiende erreicht worden ist und keine weiteren Datenbytes gelesen werden konnten (cbRead=0). Dieser Ausgang wird nicht gesetzt, wenn noch Datenbytes gelesen werden konnten (cbRead>0).

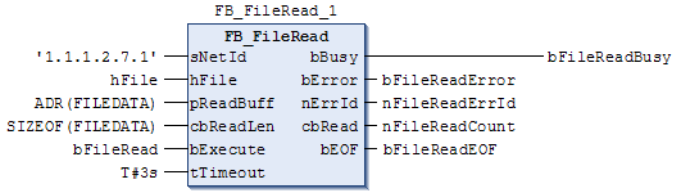
Befehlerspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70A	No memory for read buffer.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in ST:

- Dateizugriff aus der SPS [▶ 147]

Beispiel für den Aufruf des Bausteins in FBD:

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

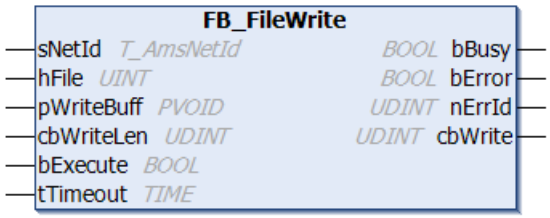


Nach steigender Flanke von bExecute und erfolgreicher Ausführung des Lese-Befehls, befinden sich in FILEDATA die aktuell gelesenen Bytes der Datei. Wie viele Bytes beim jeweils letzten Lesevorgang tatsächlich gelesen wurden, kann anhand des Parameters cbRead ermittelt werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.8 FB_FileWrite



Der Funktionsbaustein FB_FileWrite schreibt Daten in eine Datei. Die Datei muss für einen Schreibzugriff zuvor mit dem entsprechenden Modus für Schreibzugriff geöffnet worden sein und zur weiteren Verarbeitung durch externe Programme wieder geschlossen werden.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
pWriteBuff	PVOID	Adresse des Puffers, der die zu schreibenden Daten enthält. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR-Operator ermittelt werden kann.
cbWriteLen	UDINT	Anzahl der zu schreibenden Bytes
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
cbWrite	UDINT	Enthält die Anzahl der zuletzt erfolgreich geschriebenen Datenbytes. Beim Schreibfehler ist die Anzahl der erfolgreich geschriebenen Datenbytes kleiner als die angeforderte Länge (cbWriteLen) oder Null. Ein Schreibfehler kann z. B. dann auftreten wenn der Datenträger voll ist. Beim Schreibfehler wird der bError- und nErrID-Ausgang nicht gesetzt. Da die SPS-Applikation die Anzahl der zu schreibenden Datenbytes kennt, kann sie die tatsächlich geschriebene Länge mit der angeforderten Länge vergleichen und so Schreibfehler erkennen. Beim Schreibfehler hat der interne Dateizeiger eine undefinierte Position.

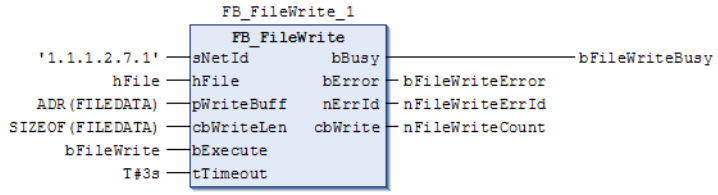
Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in ST:

- Dateizugriff aus der SPS [▶ 147]

Beispiel für den Aufruf des Bausteins in FBD:

```
PROGRAM Test
VAR
    fbFileWrite       : FB_FileWrite;
    hFile             : UINT;
    bFileWrite        : BOOL;
    bFileWriteBusy    : BOOL;
    bFileWriteError   : BOOL;
    nFileWriteErrorId: UDINT;
    nFileWriteCount   : UDINT;
    fileData          : ARRAY[0..9] OF BYTE;
END_VAR
```



Im Beispiel werden, nach steigender Flanke von bExecute, 10 Byte des Arrays fileData in die Datei geschrieben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.9 FB_FileSeek

Der Funktionsbaustein FB_FileSeek setzt den Dateizeiger einer geöffneten Datei auf eine definierbare Position.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
nSeekPos	DINT	Neue Position des Dateizeigers
eOrigin	E_SeekOrigin	Relative Position, zu der der Dateizeiger bewegt werden soll (Typ: E_SeekOrigin [▶ 126]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

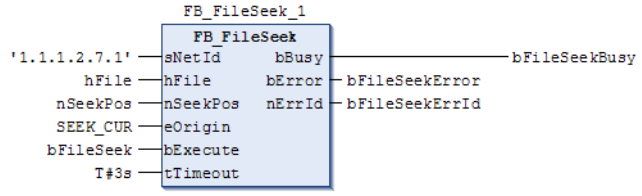
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in FBD:

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



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.10 FB_FileTell

Der Funktionsbaustein FB_FileTell ermittelt die aktuelle Position des Dateizeigers. Die Position gibt den relativen Offset zum Dateianfang an.

Beachten Sie, dass bei Dateien, die in dem Modus „Anhängen am Ende der Datei“ geöffnet wurden, die aktuelle Position durch die letzte I/O-Operation bestimmt wird und nicht dadurch, wo der nächste Schreibzugriff stattfinden wird. Wenn z.B. zuletzt gelesen wurde, dann steht der Dateizeiger an der Position, wo der nächste Lesezugriff stattfindet und nicht an der Position, wo der nächste Schreibzugriff stattfinden wird. Im Modus zum Anhängen wird der Dateizeiger vor der Schreiboperation immer an das Ende bewegt.

Wenn bisher keine I/O-Operation durchgeführt wurde und die Datei im Modus zum Anhängen geöffnet wurde, dann steht der Dateizeiger am Dateianfang.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
hFile	UINT	Datei-Handle, welches beim Aufruf des Funktionsbausteins FB_FileOpen erzeugt wurde.
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

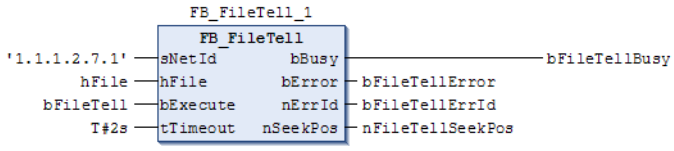
```
VAR_OUTPUT
    bBusy    : BOOL;
    bError   : BOOL;
    nErrId   : UDINT;
    nSeekPos : DINT;(* On error, nSEEKPOS returns -1 *)
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.
nSeekPos	DINT	Liefert die aktuelle Position des Dateizeigers.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x703	Invalid or unknown file handle.
0x70E	File was opened with wrong method (e.g. with obsolete FILEOPEN function block).

Beispiel für den Aufruf des Bausteins in FBD:

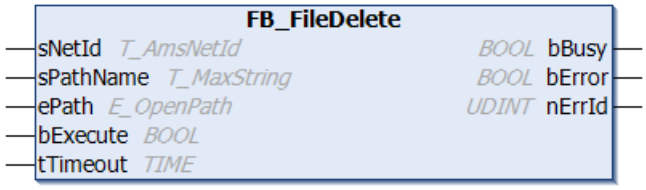
```
PROGRAM Test
VAR
    fbFileTell         : FB_FileTell;
    hFile              : UINT;
    bFileTell          : BOOL;
    bFileTellBusy      : BOOL;
    bFileTellError     : BOOL;
    nFileTellErrorId   : UDINT;
    nFileTellSeekPos   : DINT;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.11 FB_FileDelete



Der Funktionsbaustein FB_FileDelete löscht eine Datei auf dem Datenträger.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
sPathName	T_MaxString	Dateiname mit dem gesamten Pfad (Typ: T_MaxString [▶ 132]).
ePath	E_OpenPath	Über diesen Eingang kann ein TwinCAT-Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden (Typ: E_OpenPath [▶ 126]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

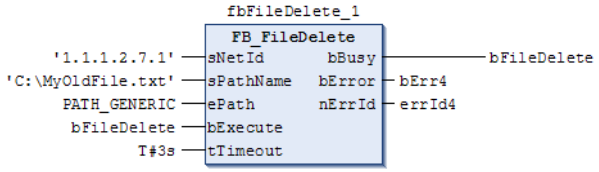
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x70C	File not found. Invalid sPathName or ePath parameter.

Beispiel für den Aufruf des Bausteins in FBD:

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



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.12 FB_FileRename

Mit diesem Funktionsbaustein FB_FileRename kann eine Datei umbenannt werden.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
sOldName	T_MaxString	Alter Dateiname (Typ: T_MaxString [▶ 132])
sNewName	T_MaxString	Neuer Dateiname (Typ: T_MaxString [▶ 132])
ePath	E_OpenPath	Über diesen Eingang kann ein TwinCAT-Systempfad auf dem Zielgerät zum Öffnen der Datei angewählt werden (Typ: E_OpenPath [▶ 126]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

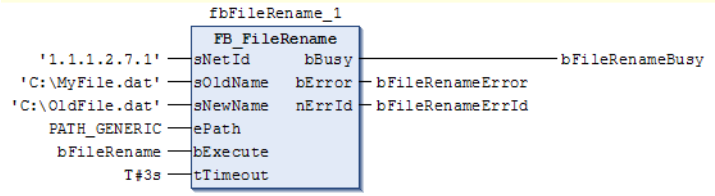
```
VAR_OUTPUT
    bBusy  : BOOL;
    bError : BOOL;
    nErrId : UDINT;
END_VAR
```

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x70C	File not found. Invalid sOldName, sNewName or ePath parameter.

Beispiel für den Aufruf des Bausteins in FBD:

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



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.13 FB_CreateDir

Mit diesem Funktionsbaustein FB_CreateDir können neue Verzeichnisse auf dem Datenträger erstellt werden.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
sPathName	T_MaxString	Name des neuen Verzeichnisses. Beim Aufruf des Bausteins kann nur ein neues Verzeichnis erstellt werden (Typ: T_MaxString [▶ 132]).
ePath	E_OpenPath	Über diesen Eingang kann ein TwinCAT-Systempfad für das neue Verzeichnis auf dem Zielgerät angewählt werden (Typ: E_OpenPath [▶ 126]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x723	Folder is already existing or invalid sPathName or ePath parameter.

Beispiel in ST:

Bei einer steigenden Flanke am bCreate wird im Hauptverzeichnis C:\ ein neues Verzeichnis mit dem Namen „PRJDATA“ erstellt und bei steigender Flanke am bRemove kann ein Verzeichnis mit dem gleichen Namen gelöscht werden.

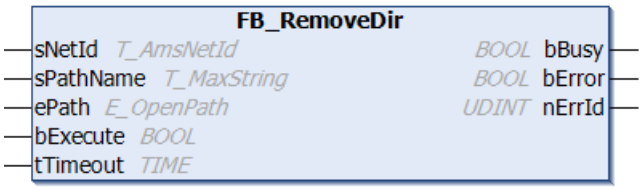
Bei bBootFolder = TRUE kann ein Verzeichnis im Verzeichnis *..\TwinCAT\Boot* erstellt oder gelöscht werden.

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

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

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

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


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

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.3.14 FB_RemoveDir



Mit diesem Funktionsbaustein FB_RemoveDir kann ein Verzeichnis vom Datenträger gelöscht werden. Ein Verzeichnis, das Dateien enthält, kann nicht gelöscht werden.

> Dieser Funktionsbaustein ist nur bedingt für ein Logging in Echtzeit geeignet.
> Für eine gute Performance empfehlen wir die Verwendung des kostenpflichtigen Produkts TF3500 TwinCAT Analytics Logger.

Eingänge

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

Name	Typ	Beschreibung
sNetId	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
sPathName	T_MaxString	Zu löschender Verzeichnisname. Beim Aufruf des Funktionsbausteins kann nur ein Verzeichnis gelöscht werden. Die letzte Komponente von sPathName muss den zu löschenden Verzeichnisnamen beinhalten (Typ: T_MaxString [▶ 132]).
ePath	E_OpenPath	Über diesen Eingang kann ein TwinCAT-Systempfad zum Löschen des Verzeichnisses auf dem Zielgerät angewählt werden (Typ: E_OpenPath [▶ 126]).
bExecute	BOOL	Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.
tTimeout	TIME	Gibt die Timeout-Zeit an, die bei der Ausführung des ADS-Kommandos nicht überschritten werden darf.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl erzeugt werden.
bError	BOOL	Wenn bei der Ausführung des Befehls ein Fehler auftritt, wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.
nErrId	UDINT	Liefert bei einem gesetzten bError-Ausgang den ADS-Fehlercode [▶ 152] oder den befehlsspezifischen Fehlercode.

Befehlsspezifischer Fehlercode	Mögliche Ursache
0x70C	Folder not found or invalid sPathName or ePath parameter.

Beispiel in ST:

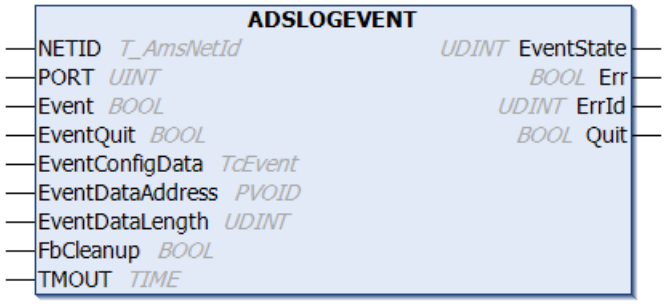
Siehe Beschreibung von FB_CreateDir [▶ 69].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.4 EventLogger-Funktionsbausteine

3.4.1 ADSLOGEVENT



Der Funktionsbaustein erlaubt das Absenden und Quittieren von Meldungen zum TwinCAT EventLogger.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

Der TwinCAT EventLogger wurde durch den Nachfolger TwinCAT 3 EventLogger abgelöst. Der ältere TwinCAT EventLogger wird von TwinCAT 3 bis zur Version 3.1.4024 unterstützt. Neuere TwinCAT-Versionen (>= 3.1.4026.0) unterstützen nur den neueren TwinCAT 3 EventLogger. SPS-Funktionsbausteine hierzu befinden sich in der SPS Bibliothek Tc3_EventLogger.

Eingänge

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

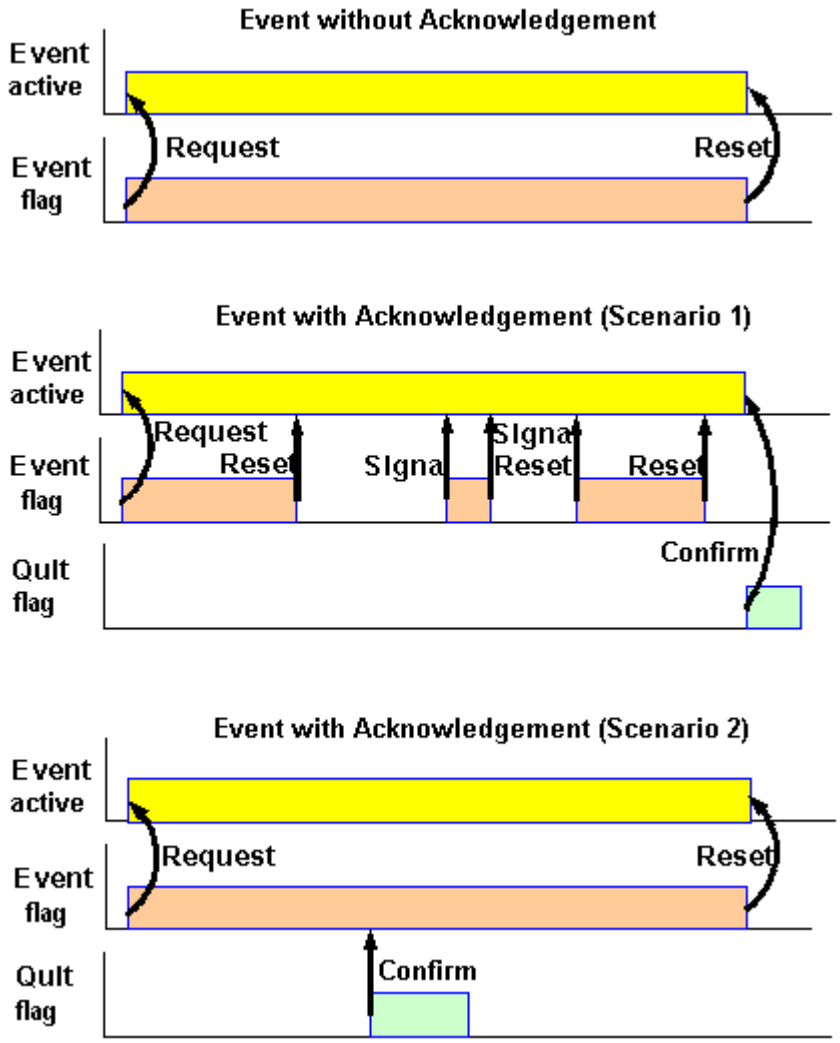
Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
PORT	T_AmsPort	Portnummer des ADS-Gerätes. Der TwinCAT EventLogger hat die Portnummer 110 (Typ: T_AmsPort [▶ 130]).
Event	BOOL	Mit der steigenden Flanke wird das „Kommen“ des Events signalisiert, mit der fallenden Flanke das „Gehen“ des Events.
EventQuit	BOOL	Mit der steigenden Flanke wird das Event quittiert.
EventConfig Data	TcEvent	Datenstruktur mit den Event-Parametern (Typ: TcEvent [▶ 133]).
EventData Address	PVOID	Adresse mit den Daten, die mit dem Event geschickt werden sollen.
EventData Length	UDINT	Länge der Daten, die mit dem Event geschickt werden sollen.
FbCleanup	BOOL	Bei TRUE wird der Baustein komplett initialisiert.
TMOUT	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

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

Name	Typ	Beschreibung
EventState	UDINT	Zustand des Events.
Err	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ErrId enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist Err = TRUE und ErrId = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
ErrId	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.
Quit	BOOL	Quittiert das Event.

Meldungen quittieren



Das obere Bild stellt den prinzipiellen Ablauf dar.

Bei nicht quittierungspflichtigen Meldungen wird mit der steigenden Flanke am Event-Eingang des Bausteins das Event gemeldet und damit aktiv im Eventlogger. Die fallende Flanke am Event-Eingang löst den Reset aus. Mit diesem Signal wird das Event im EventLogger wieder abgemeldet.

Bei quittierungspflichtigen Meldungen wird das Event wieder mit der steigenden Flanke am Event-Eingang aktiviert. Deaktiviert wird das Event entweder

- durch die fallende Flanke am Event-Eingang (wenn vorher ein Quittierungssignal aus der SPS mit dem Quit-Eingang oder von der Visualisierung gekommen ist) oder
- durch die steigende Flanke am Quit-Eingang (wenn vorher ein Reset durch eine fallende Flanke am Event-Eingang ausgelöst wurde).

Wenn zwischen Event-Aktivierung und dem Ankommen der Quittierung ein Reset des Events kommt, heißt das nächste Ankommen des Event-Eingangs „Signal“. Damit wird ein Request bei bereits aktiven Events gemeldet.

Beispiel für den Aufruf des Bausteins in ST:

- Eventlogger Meldungen aus der SPS senden/quittieren [▶ 146]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC oder CX (x86, x64, ARM)	Tc2_System (System)

3.4.2 ADSCLEAREVENTS

Der Funktionsbaustein sendet und quittiert Meldungen zum TwinCAT EventLogger.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

Der TwinCAT EventLogger wurde durch den Nachfolger TwinCAT 3 EventLogger abgelöst. Der ältere TwinCAT EventLogger wird von TwinCAT 3 bis zur Version 3.1.4024 unterstützt. Neuere TwinCAT-Versionen (>= 3.1.4026.0) unterstützen nur den neueren TwinCAT 3 EventLogger. SPS-Funktionsbausteine hierzu befinden sich in der SPS Bibliothek Tc3_EventLogger.

Eingänge

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

Name	Typ	Beschreibung
NETID	T_AmsNetId	String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird (Typ: T_AmsNetId [▶ 130]).
bClear	BOOL	Mit der steigenden Flanke werden die Events gelöscht.
iMode	UDINT	Modus zum Löschen der Events. Definiert im Enum E_TcEventClearModes [▶ 127].
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

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

Name	Typ	Beschreibung
bBusy	BOOL	Bei der Aktivierung des Funktionsbausteins wird dieser Ausgang auf TRUE gesetzt und bleibt gesetzt, bis eine Rückmeldung erfolgt. Solange bBusy = TRUE ist, kann kein neuer Befehl ausgeführt werden.
bErr	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in iErrId enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist bErr = TRUE und iErrId = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
iErrId	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC oder CX (x86, x64, ARM)	Tc2_System (System)

3.4.3 FB_SimpleAdsLogEvent

Der Funktionsbaustein erlaubt das Absenden und Quittieren von Meldungen zum TwinCAT EventLogger. Im Gegensatz zum Baustein ADSLOGEVENT können Events mit dem Baustein FB_SimpleAdsLogEvent aus der PLC heraus nicht parametriert werden, jedoch lassen sich die Events auf einfache Weise setzen, zurücksetzen und quittieren.

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

Der TwinCAT EventLogger wurde durch den Nachfolger TwinCAT 3 EventLogger abgelöst. Der ältere TwinCAT EventLogger wird von TwinCAT 3 bis zur Version 3.1.4024 unterstützt. Neuere TwinCAT-Versionen (>= 3.1.4026.0) unterstützen nur den neueren TwinCAT 3 EventLogger. SPS-Funktionsbausteine hierzu befinden sich in der SPS Bibliothek Tc3_EventLogger.

Eingänge

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

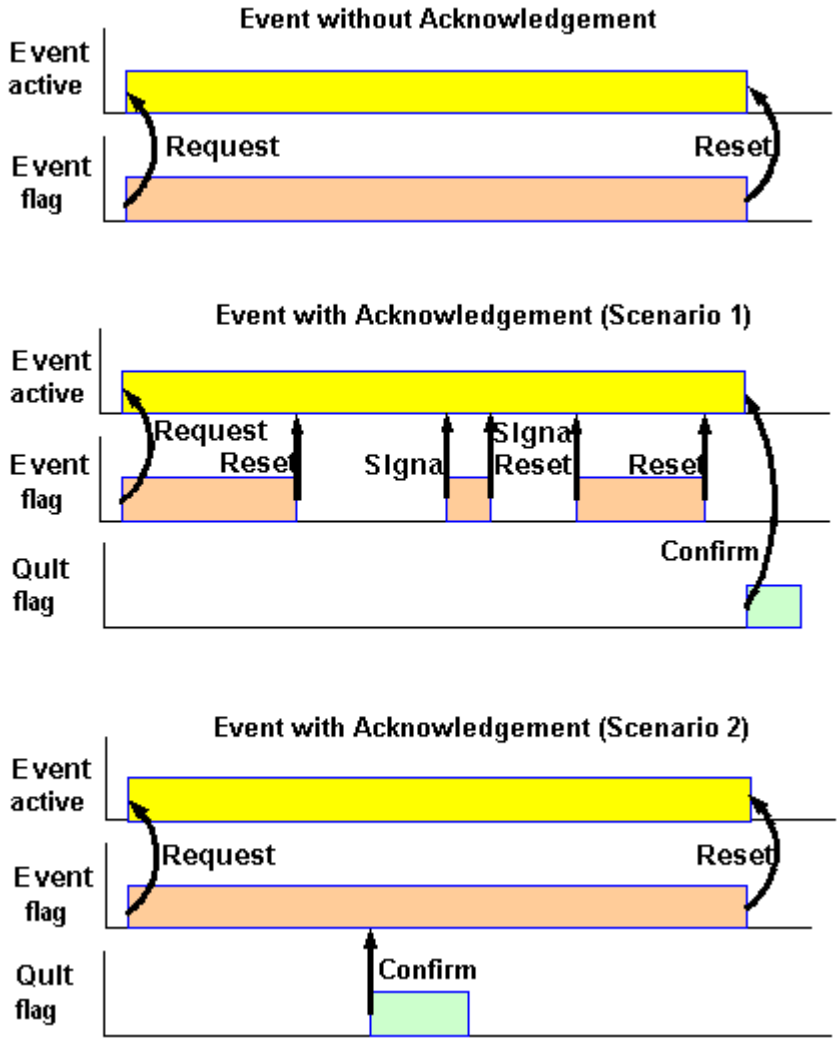
Name	Typ	Beschreibung
SourceId	INT	ID der Source. Wird zur eindeutigen Identifizierung der Source im EventLogger genutzt.
EventId	INT	ID des Events. Wird zur eindeutigen Identifizierung des Events im EventLogger genutzt.
bSetEvent	BOOL	Mit der steigenden Flanke wird das „Kommen“ des Events signalisiert, mit der fallenden Flanke das „Gehen“ des Events.
bQuit	BOOL	Mit der steigenden Flanke wird das Event quittiert.

Ausgänge

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

Name	Typ	Beschreibung
ErrId	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.
Error	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in ErrId enthalten. Wenn der Baustein einen Timeout-Fehler hat, so ist Error = TRUE und ErrId = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

Meldungen quittieren



Das obere Bild stellt den prinzipiellen Ablauf dar.

Bei nicht quittierungspflichtigen Meldungen wird mit der steigenden Flanke am Event-Eingang des Bausteins das Event gemeldet und damit im EventLogger aktiv. Die fallende Flanke am Event-Eingang löst den Reset aus. Mit diesem Signal wird das Event im EventLogger wieder abgemeldet.

Bei quittierungspflichtigen Meldungen wird das Event wieder mit der steigenden Flanke am Event-Eingang aktiviert. Deaktiviert wird das Event entweder

- durch die fallende Flanke am Event-Eingang (wenn vorher ein Quittierungssignal aus der SPS mit dem Quit-Eingang oder von der Visualisierung gekommen ist) oder

- durch die steigende Flanke am Quit-Eingang (wenn vorher ein Reset durch eine fallende Flanke am Event-Eingang ausgelöst wurde).

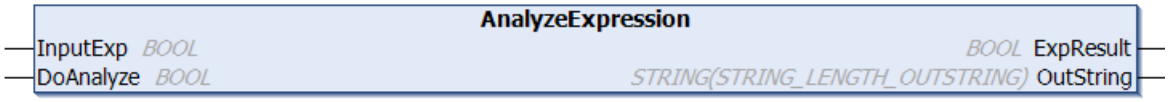
Wenn zwischen Event-Aktivierung und Ankommen der Quittierung ein Reset des Event kommt, heißt das nächste Ankommen des Event-Eingangs „Signal“. Damit wird ein Request bei bereits aktiven Events gemeldet.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC oder CX (x86, x64, ARM)	Tc2_System (System)

3.5 IEC-Schritt-/ SFC-Flags-Funktionsbausteine

3.5.1 AnalyzeExpression



Der Funktionsbaustein kann in SPS-Projekten genutzt werden, die die AS-Flags benutzen. Es werden keine Instanzen erzeugt. Im Projekt muss nur die entsprechende SPS-Bibliothek eingebunden werden. Weitere Konfigurationsvoraussetzungen finden Sie in den nachfolgenden Erläuterungen.

Die Bausteine AnalyzeExpression und AnalyzeExpressionTable können zur Analyse von Transitionen bzw. Weiterschaltbedingungen verwendet werden. Wenn eine Transition nach einer eingestellten Zeit nicht schaltet, kann diese Transition mithilfe der Bausteine analysiert werden.

Mithilfe der Bausteine können Sie nur Ausdrücke bzw. Transitionen analysieren, die in der Programmiersprache ST implementiert sind.

- AnalyzeExpression:
  - Der Baustein gibt das Analyseergebnis, warum nicht weitergeschaltet wird (d. h. welche Teilbedingung/en nicht erfüllt ist/sind), gebündelt in einem STRING aus. Die Variablen, die die Teilbedingungen bilden, werden per Operator miteinander verknüpft (z. B. `bVar1 AND (bVar2 OR bVar3)`).
  - Die Ausgabe erfolgt in dem AS-Flag „SFCErrorAnalyzation“.
- AnalyzeExpressionTable:
  - Der Baustein gibt alle nicht-schaltenden Variablen einzeln aus. Die einzelnen Variablen werden dabei jeweils als Arrayelement erfasst bzw. ausgegeben. Zu den gelisteten Informationen pro Arrayelement gehören der Name, die Adresse, der Kommentar und der aktuelle Wert der Variablen.
  - Die Ausgabe erfolgt in dem AS-Flag „SFCErrorAnalyzationTable“.

Konfigurationsvoraussetzungen

Um AnalyzeExpression bzw. AnalyzeExpressionTable für AS zu aktivieren, sind die folgenden Einstellungen vorzunehmen:

- Binden Sie die SPS-Bibliothek Tc2_System ein.
- Deklarieren Sie in der AS-POU die folgende Variable:
  `SFCEnableLimit : BOOL := TRUE;`

- Konfigurieren Sie für den Schritt bzw. die Schritte des AS-Diagramms, dessen/deren nachfolgende Transition/Weiterschaltbedingung analysiert werden soll, im Eigenschaftenfenster jeweils eine maximale Aktiv-Zeit (siehe auch AS-Elementeigenschaften).
- Konfigurieren Sie die AS-Einstellungen in den SPS-Projekteigenschaften oder in den POU-Eigenschaften
  (siehe auch AS-Flags und Befehl Eigenschaften (SPS-Projekt) > Kategorie AS):
  - Registerkarte **Flags**:
    Aktivieren Sie für die folgenden AS-Flags die Auswahlkästchen **Aktiv** und **Deklarieren**:
    SFCError, SFCEnableLimit, SFCErrorAnalyzation, SFCErrorAnalyzationTable
  - Registerkarte **Übersetzen**:
    Aktivieren Sie die Option **Nur aktive Transitionen berechnen**.

Beispiel

In dem nachfolgenden Beispiel wurden die oben genannten Konfigurationen vorgenommen. Für den Schritt „Step1“ wurde die maximale Aktiv-Zeit auf 1 s gestellt. Wenn die zugehörige ausgehende Transition „Trans_ST“ nach 1 s noch nicht geschaltet hat, wird diese Transition von den Bausteinen AnalyzeExpression und AnalyzeExpressionTable analysiert. Die Variable SFCError wird auf TRUE gesetzt und die Variable SFCErrorStep bekommt den Wert 'Step 1'.

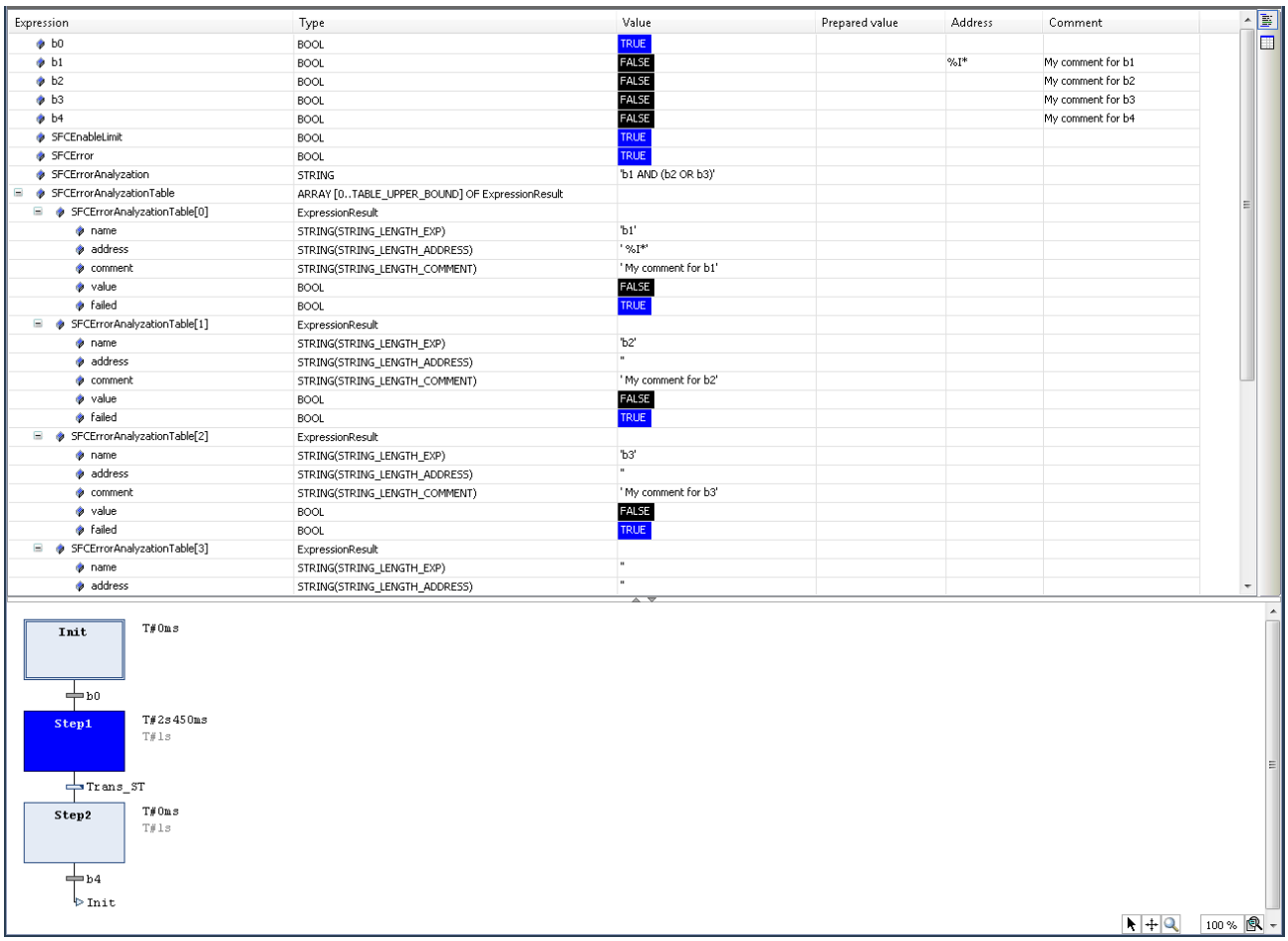
Die Analyseergebnisse „SFCErrorAnalyzation“ bzw. „SFCErrorAnalyzationTable“ zeigen an, welcher (Teil-)Ausdruck noch nicht geschaltet hat, damit der Schritt „Step1“ verlassen werden kann.

Die Transition „Trans_ST“ besteht aus dem Ausdruck

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

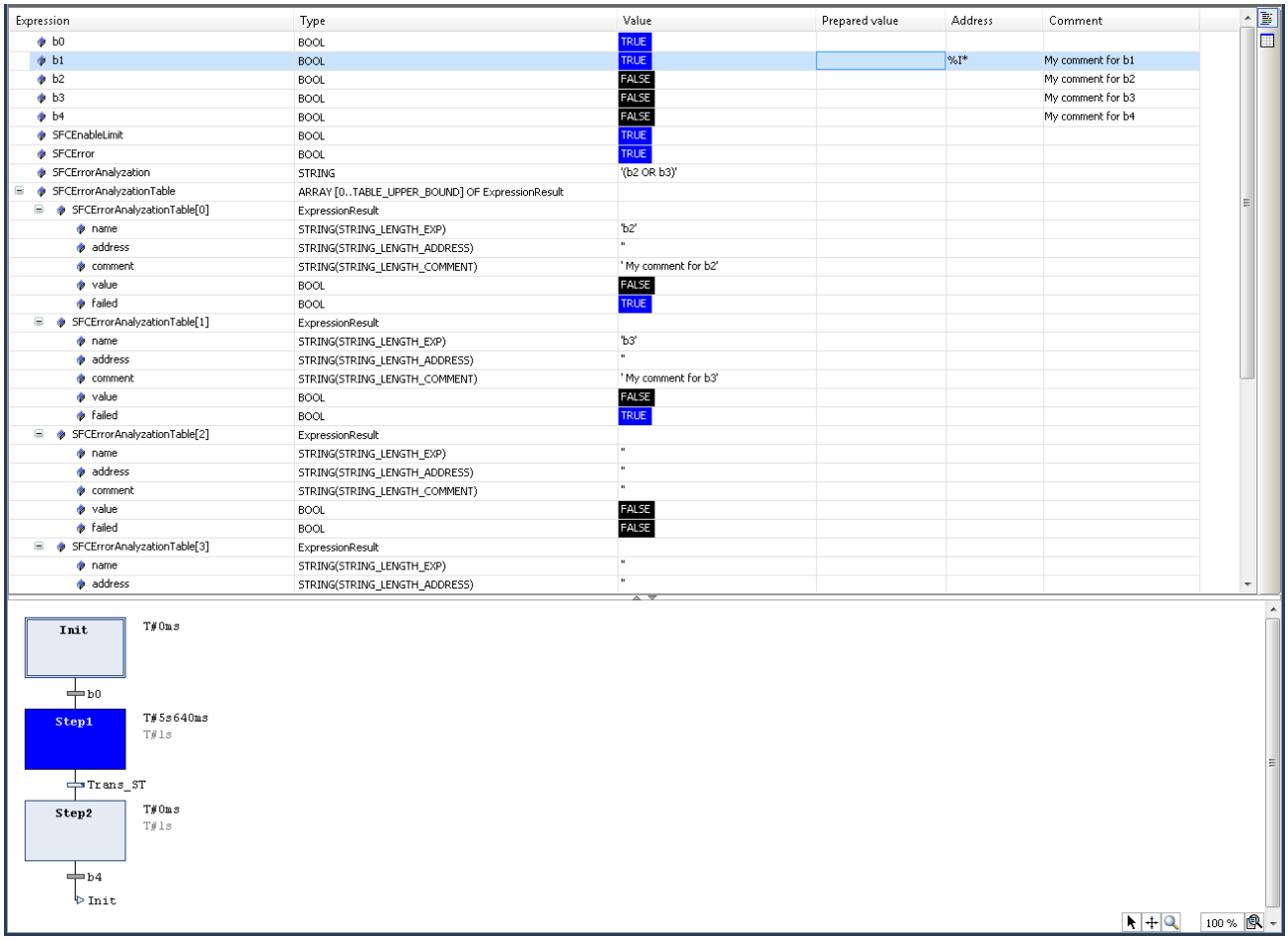
- Situation 1: Noch keine der drei Variablen b1, b2, b3 besitzt den Wert TRUE.
  - „SFCErrorAnalyzation“ zeigt das Analyseergebnis 'b1 AND (b2 OR b3)'.
  - „SFCErrorAnalyzationTable“ führt alle drei Variablen b1, b2, b3 mit detaillierten Variableninformationen auf.
  - Sehen Sie hierzu auch Abbildung 1.
- Situation 2: Variable b1 wird auf TRUE gesetzt. Die Analyseergebnisse verändern sich entsprechend.
  - „SFCErrorAnalyzation“ zeigt das Analyseergebnis '(b2 OR b3)'.
  - „SFCErrorAnalyzationTable“ führt nur noch die beiden Variablen b2 und b3 mit den entsprechenden Variableninformationen auf.
  - Sehen Sie hierzu auch Abbildung 2.

Abbildung 1:



Expression	Type	Value	Prepared value	Address	Comment
b0	BOOL	TRUE			
b1	BOOL	FALSE		%I*	My comment for b1
b2	BOOL	FALSE			My comment for b2
b3	BOOL	FALSE			My comment for b3
b4	BOOL	FALSE			My comment for b4
SFCEnableLimit	BOOL	TRUE			
SFCError	BOOL	TRUE			
SFCErrorAnalyzation	STRING	'b1 AND (b2 OR b3)'			
SFCErrorAnalyzationTable	ARRAY [0..TABLE_UPPER_BOUND] OF ExpressionResult				
SFCErrorAnalyzationTable[0]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b1'			
address	STRING(STRING_LENGTH_ADDRESS)	'%I*'			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b1'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[1]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b2'			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b2'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[2]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b3'			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b3'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[3]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	''			
address	STRING(STRING_LENGTH_ADDRESS)	''			

Abbildung 2:



Expression	Type	Value	Prepared value	Address	Comment
b0	BOOL	TRUE			
b1	BOOL	TRUE		%I*	My comment for b1
b2	BOOL	FALSE			My comment for b2
b3	BOOL	FALSE			My comment for b3
b4	BOOL	FALSE			My comment for b4
SFCEnableLimit	BOOL	TRUE			
SFCError	BOOL	TRUE			
SFCErrorAnalyzation	STRING	'(b2 OR b3)'			
SFCErrorAnalyzationTable	ARRAY [0..TABLE_UPPER_BOUND] OF ExpressionResult				
SFCErrorAnalyzationTable[0]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b2'			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b2'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[1]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b3'			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b3'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[2]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	''			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	''			
value	BOOL	FALSE			
failed	BOOL	FALSE			
SFCErrorAnalyzationTable[3]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	''			
address	STRING(STRING_LENGTH_ADDRESS)	''			

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.5.2 AnalyzeExpressionTable

Der Funktionsbaustein kann in SPS-Projekten genutzt werden, die die AS-Flags benutzen. Es werden keine Instanzen erzeugt. Im Projekt muss nur die entsprechende SPS-Bibliothek eingebunden werden. Weitere Konfigurationsvoraussetzungen finden Sie in den nachfolgenden Erläuterungen.

Die Bausteine AnalyzeExpression und AnalyzeExpressionTable können zur Analyse von Transitionen bzw. Weiterschaltbedingungen verwendet werden. Wenn eine Transition nach einer eingestellten Zeit nicht schaltet, kann diese Transition mithilfe der Bausteine analysiert werden.

> Mithilfe der Bausteine können Sie nur Ausdrücke bzw. Transitionen analysieren, die in der Programmiersprache ST implementiert sind.

- AnalyzeExpression:
  - Der Baustein gibt das Analyseergebnis, warum nicht weitergeschaltet wird (d. h. welche Teilbedingung/en nicht erfüllt ist/sind), gebündelt in einem STRING aus. Die Variablen, die die Teilbedingungen bilden, werden per Operator miteinander verknüpft (z. B. `bVar1 AND (bVar2 OR bVar3)`).
  - Die Ausgabe erfolgt in dem AS-Flag „SFCErrorAnalyzation“.
- AnalyzeExpressionTable:
  - Der Baustein gibt alle nicht-schaltenden Variablen einzeln aus. Die einzelnen Variablen werden dabei jeweils als Arrayelement erfasst bzw. ausgegeben. Zu den gelisteten Informationen pro Arrayelement gehören der Name, die Adresse, der Kommentar und der aktuelle Wert der Variablen.
  - Die Ausgabe erfolgt in dem AS-Flag „SFCErrorAnalyzationTable“.

Konfigurationsvoraussetzungen

Um AnalyzeExpression bzw. AnalyzeExpressionTable für AS zu aktivieren, sind die folgenden Einstellungen vorzunehmen:

- Binden Sie die SPS-Bibliothek Tc2_System ein.
- Deklarieren Sie in der AS-POU die folgende Variable:
  `SFCEnableLimit : BOOL := TRUE;`
- Konfigurieren Sie für den Schritt bzw. die Schritte des AS-Diagramms, dessen/deren nachfolgende Transition/Weiterschaltbedingung analysiert werden soll, im Eigenschaftenfenster jeweils eine maximale Aktiv-Zeit (siehe auch AS-Elementeigenschaften).
- Konfigurieren Sie die AS-Einstellungen in den SPS-Projekteigenschaften oder in den POU-Eigenschaften
  (siehe auch AS-Flags und Befehl Eigenschaften (SPS-Projekt) > Kategorie AS):
  - Registerkarte **Flags**:
    Aktivieren Sie für die folgenden AS-Flags die Auswahlkästchen **Aktiv** und **Deklarieren**:
    SFCError, SFCEnableLimit, SFCErrorAnalyzation, SFCErrorAnalyzationTable
  - Registerkarte **Übersetzen**:
    Aktivieren Sie die Option **Nur aktive Transitionen berechnen**.

Beispiel

In dem nachfolgenden Beispiel wurden die oben genannten Konfigurationen vorgenommen. Für den Schritt „Step1“ wurde die maximale Aktiv-Zeit auf 1 s gestellt. Wenn die zugehörige ausgehende Transition „Trans_ST“ nach 1 s noch nicht geschaltet hat, wird diese Transition von den Bausteinen AnalyzeExpression und AnalyzeExpressionTable analysiert. Die Variable SFCError wird auf TRUE gesetzt und die Variable SFCErrorStep bekommt den Wert 'Step 1'.

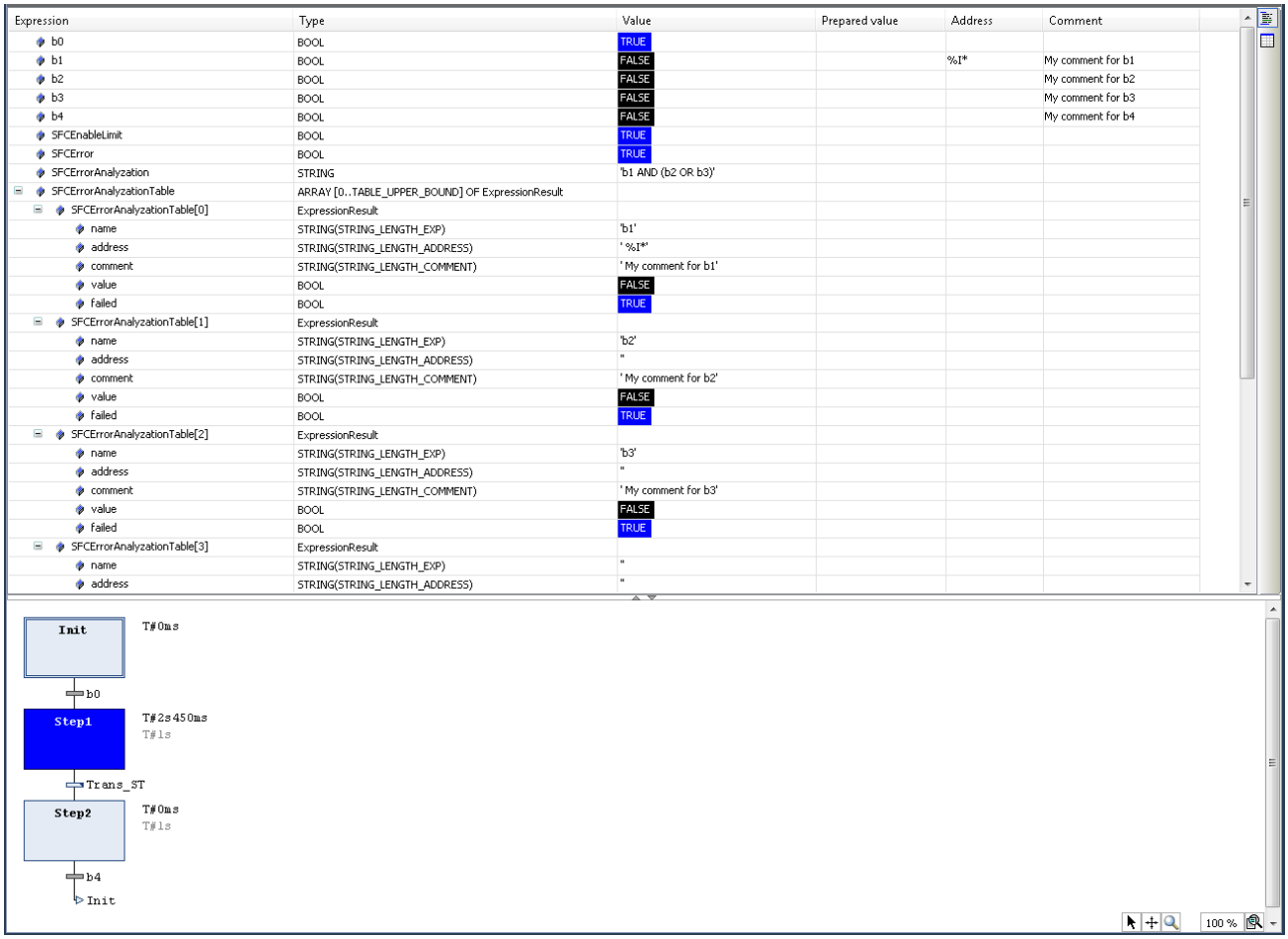
Die Analyseergebnisse „SFCErrorAnalyzation“ bzw. „SFCErrorAnalyzationTable“ zeigen an, welcher (Teil-)Ausdruck noch nicht geschaltet hat, damit der Schritt „Step1“ verlassen werden kann.

Die Transition „Trans_ST“ besteht aus dem Ausdruck

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

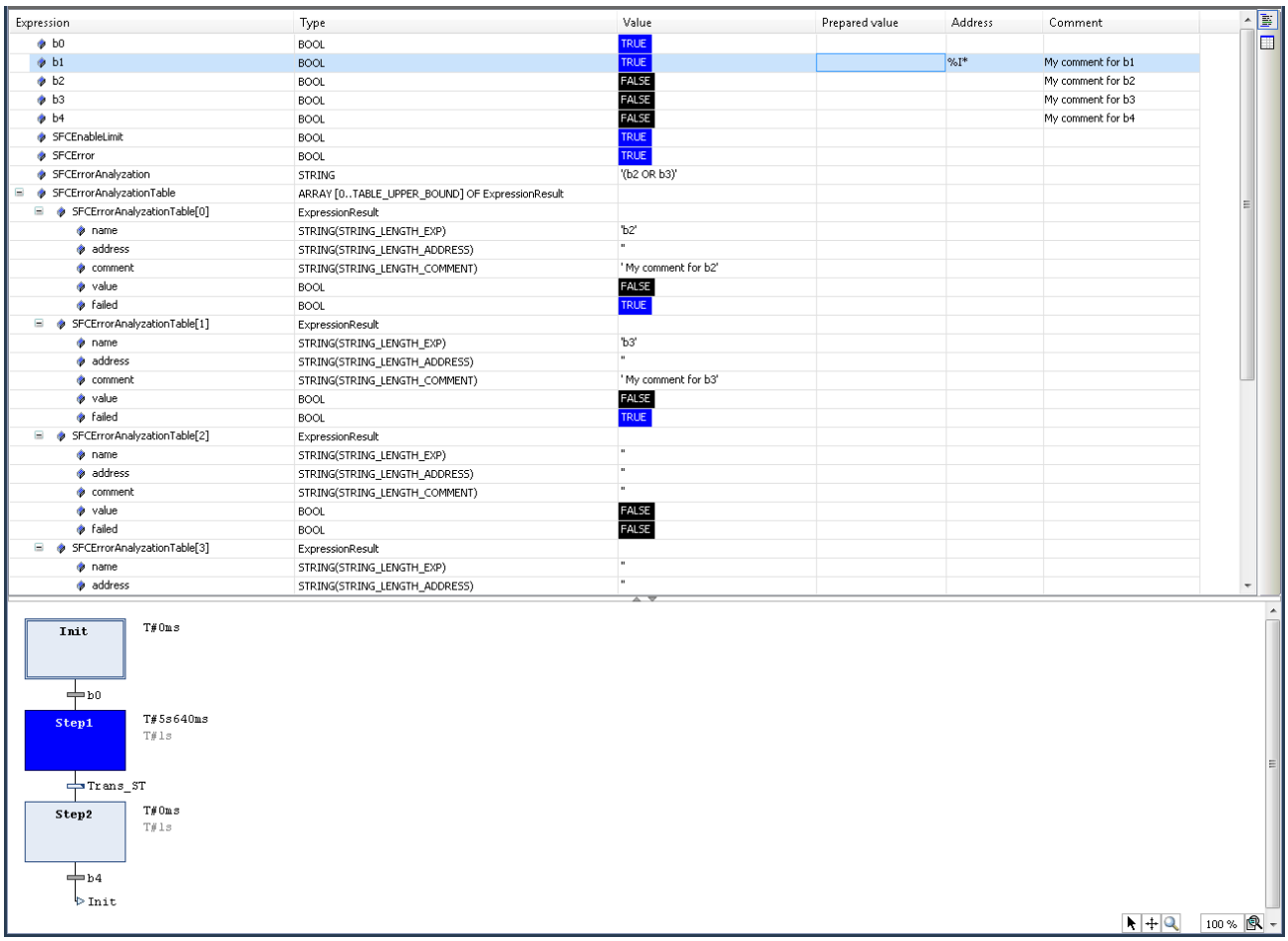
- Situation 1: Noch keine der drei Variablen b1, b2, b3 besitzt den Wert TRUE.
  - „SFCErrorAnalyzation“ zeigt das Analyseergebnis 'b1 AND (b2 OR b3)'.
  - „SFCErrorAnalyzationTable“ führt alle drei Variablen b1, b2, b3 mit detaillierten Variableninformationen auf.
  - Sehen Sie hierzu auch Abbildung 1.
- Situation 2: Variable b1 wird auf TRUE gesetzt. Die Analyseergebnisse verändern sich entsprechend.
  - „SFCErrorAnalyzation“ zeigt das Analyseergebnis '(b2 OR b3)'.
  - „SFCErrorAnalyzationTable“ führt nur noch die beiden Variablen b2 und b3 mit den entsprechenden Variableninformationen auf.
  - Sehen Sie hierzu auch Abbildung 2.

Abbildung 1:



Expression	Type	Value	Prepared value	Address	Comment
b0	BOOL	TRUE			
b1	BOOL	FALSE		%I*	My comment for b1
b2	BOOL	FALSE			My comment for b2
b3	BOOL	FALSE			My comment for b3
b4	BOOL	FALSE			My comment for b4
SFCEnableLimit	BOOL	TRUE			
SFCError	BOOL	TRUE			
SFCErrorAnalyzation	STRING	'b1 AND (b2 OR b3)'			
SFCErrorAnalyzationTable	ARRAY [0..TABLE_UPPER_BOUND] OF ExpressionResult				
SFCErrorAnalyzationTable[0]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b1'			
address	STRING(STRING_LENGTH_ADDRESS)	'%I*'			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b1'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[1]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b2'			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b2'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[2]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	'b3'			
address	STRING(STRING_LENGTH_ADDRESS)	''			
comment	STRING(STRING_LENGTH_COMMENT)	'My comment for b3'			
value	BOOL	FALSE			
failed	BOOL	TRUE			
SFCErrorAnalyzationTable[3]	ExpressionResult				
name	STRING(STRING_LENGTH_EXP)	''			
address	STRING(STRING_LENGTH_ADDRESS)	''			

Abbildung 2:



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

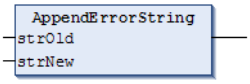
3.5.3 AnalyzeExpressionCombined

Der Funktionsbaustein wird in SPS-Projekten benötigt, die die SFC-Flags benutzen. Es werden keine Instanzen erzeugt. Im Projekt muss nur die entsprechende SPS-Bibliothek eingebunden werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.5.4 AppendErrorString

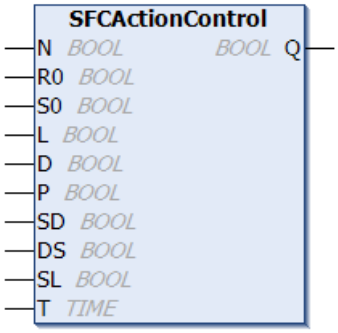


Die Funktion wird in SPS-Projekten benötigt, die die SFC-Flags benutzen. Die Funktion muss nicht im Projekt aufgerufen werden, es muss nur die entsprechende SPS-Bibliothek eingebunden werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.5.5 SFCActionControl



Dieser Funktionsbaustein wird benötigt um in SFC-Programmen/-Projekten die IEC-Schritte benutzen zu können. Im Projekt muss nur die Bibliothek mit dem FB eingebunden werden. Es werden aber keine Instanzen benötigt.

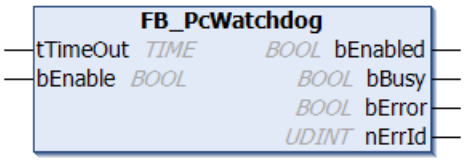
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

3.6 Watchdog-Funktionsbausteine

3.6.1 FB_PcWatchdog

Diese Funktionalität ist nur verfügbar auf den IPCs mit den Mainboards IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051.



Der Funktionsbaustein FB_PcWatchdog aktiviert einen Hardware-Watchdog auf dem PC. Der Watchdog wird über bEnable = TRUE und die Timeout-Zeit aktiviert. Die Timeoutzeit kann minimal 1 s und maximal 255 s sein. Aktiviert wird der Watchdog über bEnable = TRUE und tTimeOut >=1 s.

Wenn der Watchdog einmal aktiviert wurde, muss der Funktionsbaustein zyklisch in kürzeren Abständen aufgerufen werden als tTimeOut, da bei Ablauf der tTimeOut-Zeit der PC automatisch einen Neustart durchführt. Der Watchdog kann daher dafür eingesetzt werden um Systeme automatisch neu zu booten, die in eine Endlosschleife gelaufen sind bzw. bei denen die PLC steht.

Der Watchdog kann über bEnable = FALSE bzw. die tTimeOut-Zeit = 0 deaktiviert werden.

HINWEIS

Reboot des PCs

Der Watchdog muss vor der Verwendung von Breakpoints, einem SPS-Reset bzw. Urlöschen und vor einem TwinCAT-Stopp, einem Wechsel in den Konfig-Modus oder dem Aktivieren der Konfiguration deaktiviert werden, da es sonst unmittelbar zum Reboot des PCs nach Ablauf der Timeout-Zeit kommt.

Eingänge

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

Name	Typ	Beschreibung
tTimeOut	TIME	Watchdog-Zeit, nach deren Ablauf ein Neustart durchgeführt wird.
bEnable	BOOL	Aktivieren bzw. Deaktivieren des Watchdogs.

Ausgänge

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

Name	Typ	Beschreibung
bEnabled	BOOL	TRUE = Watchdog aktiviert, FALSE = Watchdog deaktiviert
bBusy	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt.
bError	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in nErrId enthalten. Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
nErrId	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Beispiel für den Aufruf des Bausteins in ST:

```
PROGRAM MAIN
VAR
    fbPcWatchDog  : FB_PcWatchdog;
    tWDTime       : TIME := T#10s;
    bEnableWD     : BOOL;
    bWDActive     : BOOL;
END_VAR

IF bEnableWD OR bWDActive THEN
    fbPcWatchDog(tTimeOut := tWDTime, bEnable := bEnableWD);
    bWDActive := fbPcWatchDog.bEnabled;
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	IPCs mit den Mainboards IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051	PLC Lib Tc2_System

3.6.2 FB_PcWatchDog_BAPI

Diese Funktionalität ist nur verfügbar auf IPCs und Embedded-PCs mit einer BIOS-Version, die die BIOS-API unterstützt.

Der Funktionsbaustein FB_PcWatchdog_BAPI aktiviert einen Hardware-Watchdog auf dem PC. Der Watchdog wird über bExecute = TRUE und die Watchdogzeit aktiviert. Die Watchdogzeit kann minimal 1 s und maximal 15300 s (255 min) sein. Aktiviert wird der Watchdog über bEnable = TRUE und nWatchdogTimeS >=1 s.

Wenn der Watchdog einmal aktiviert wurde, muss der Funktionsbaustein zyklisch in kürzeren Abständen aufgerufen werden als nWatchdogTimeS, da bei Ablauf der nWatchdogTimeS-Zeit der PC automatisch einen Neustart durchführt. Der Watchdog kann daher dafür eingesetzt werden um Systeme automatisch neu zu booten, die in eine Endlosschleife gelaufen sind bzw. bei denen die SPS steht.

HINWEIS

Reboot des PCs

Der Watchdog muss vor der Verwendung von Breakpoints, einem SPS-Reset bzw. Urlöschen und vor einem TwinCAT-Stopp, einem Wechsel in den Konfig-Modus oder dem Aktivieren der Konfiguration deaktiviert werden, da es sonst unmittelbar nach Ablauf der Zeit nWatchdogTimeS zum Reboot des PCs kommt.

Eingänge

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

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netwerkkennung des Geräts (leerer String oder lokale Netzwerkkennung)
nWatchdogTimeS	UDINT	Watchdogzeit in Sekunden, 0 = deaktiviert, >0 (max. 15300) = aktiviert.
bExecute	BOOL	Mit steigender Flanke wird das Kommando ausgeführt. Der Eingang muss zurückgesetzt werden, sobald der Baustein nicht mehr aktiv ist (`bBusy=FALSE`).
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der internen ADS-Kommunikation an.

Ausgänge

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

Name	Typ	Beschreibung
bEnabled	BOOL	TRUE = Watchdog aktiviert, FALSE = Watchdog deaktiviert
bBusy	BOOL	Dieser Ausgang bleibt so lange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt.
bError	BOOL	Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in nErrId enthalten. Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.
nErrId	UDINT	ADS-Fehlercode [▶ 152] oder befehlsspezifischer Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Beispiel für den Aufruf des Bausteins in ST:

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

```
fbTimer();

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

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

        fbWatchdog(bExecute := FALSE);

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

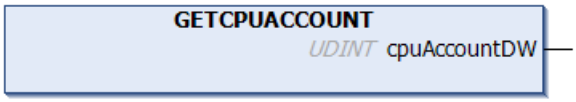
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	IPCs und Embedded-PCs mit einer BIOS-Version, die die BIOS-API unterstützt.	PLC Lib Tc2_System-Version >=3.4.14.0

3.7 Zeitfunktionsbausteine

3.7.1 GETCPUACCOUNT

Diese Funktionalität ist in dem SPS-Laufzeitsystem auf einer Windows-CE-Plattform nicht verfügbar.



Mit diesem Funktionsbaustein kann der Zyklusticker einer SPS-Task ausgelesen werden. Der Zyklusticker der SPS-Task wird nur während der Ausführungszeit der Task inkrementiert. Der Zählwert ist ein 32-Bit-Integer-Wert, der, unabhängig von internen Taktrate der CPU, in 100ns-Ticks umgerechnet ausgegeben wird. Der Zählwert wird, auf 100 ns genau, bei jedem Aufruf der SPS-Task aufgefrischt und kann z. B. für Timing-Aufgaben eingesetzt werden. Eine Einheit entspricht 100 ns.

Eingänge

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

Ausgänge

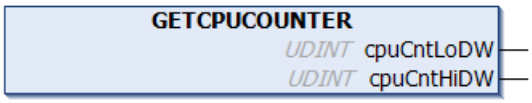
```
VAR_OUTPUT
    cpuAccountDW : UDINT;
END_VAR
```

Name	Typ	Beschreibung
cpuAccountDW	UDINT	Aktueller Wert des Tickers der SPS-Task

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC or CX (x86, x64)	Tc2_System (System)

3.7.2 GETCPUCOUNTER



Mit diesem Funktionsbaustein kann der Zyklusticker der CPU ausgelesen werden. Der Zählwert ist ein relativer, 64-Bit-Integer-Wert, der, unabhängig von der internen Taktrate der CPU, in 100-ns-Ticks umgerechnet ausgegeben wird. Der Zählwert wird, auf 100 ns genau, bei jedem Aufruf durch das SPS-System aufgefrischt und kann z. B. für Timing-Aufgaben eingesetzt werden. Eine Einheit entspricht 100 ns. Der Grund, warum dieser Dienst als Baustein und nicht als Funktion implementiert ist, ergibt sich lediglich aus der Tatsache, dass zwei Werte zurückgeliefert werden müssen und eine Funktion dieses definitionsgemäß nicht leisten kann.

Eingänge

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

Ausgänge

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

Name	Typ	Beschreibung
cpuCntLoDW	UDINT	Niederwertige 4 Byte des Zählwerts
cpuCntHiDW	UDINT	Höherwertige 4 Byte des Zählwerts

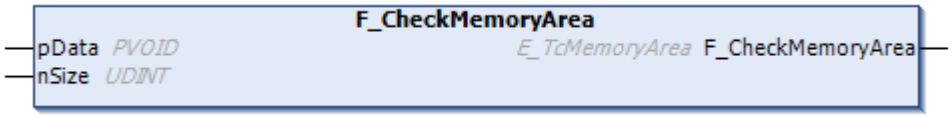
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4 Funktionen

4.1 Generelle Funktionen

4.1.1 F_CheckMemoryArea



Die Funktion liefert Informationen darüber, in welchem Speicherbereich sich die angefragte Variable mit angegebener Größe befindet. Hierzu wird ein Rückgabewert vom Typ E_TcMemoryArea [▶ 128] verwendet.

FUNCTION F_CheckMemoryArea: E_TcMemoryArea

Eingänge

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

Name	Typ	Beschreibung
pData	PVOID	Speicheradresse der Variablen
nSize	UDINT	Größe der Variablen in Bytes

Beispiel

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

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

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

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

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4022	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.2 F_CmpLibVersion

Die Funktion F_CmpLibVersion vergleicht die Version einer existierenden Bibliothek mit der Version, die benötigt wird. Jede Bibliothek besitzt eine eigene Versionsinformation als Konstante vom Typ: ST_LibVersion. Der Name der Konstanten hat den Format: stLibVersion_Bibliotheksname.

FUNCTION F_CmpLibVersion: DINT

Eingänge

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

Name	Typ	Beschreibung
stVersion	ST_LibVersion	Version der existierenden Bibliothek (Typ: ST_LibVersion)
iMajor	UINT	Benötigte Hauptnummer (major number)
iMinor	UINT	Benötigte Unternummer (minor number)
iBuild	UINT	Benötigte Build-Nummer
iRevision	UINT	Benötigte Revisionsnummer

Rückgabeparameter	Verhältnis der Versionen
-1	Die Version, die Sie haben, ist kleiner als benötigt.
0	Die Version, die Sie haben, ist die benötigte Version.
+1	Die Version, die Sie haben, ist höher als benötigt.

Beispiel in ST:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.3 F_CreateIPv4Addr

Die Funktion generiert eine formatierte (IPv4) Internet Protocol-Netzwerkadresse und liefert diese als Rückgabeparameter vom Typ String zurück (z. B. 172.16.7.199).

FUNCTION F_CreateIPv4Addr : T_IPv4Addr

Eingänge

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

Name	Typ	Beschreibung
nIds	T_IPv4AddrArr	Byte-Array: Jedes Byte entspricht einem Adressbyte der (IPv4) Internet-Protocol-Netzwerkadresse. Die Adressbytes haben die Netzwerk-Byte-Reihenfolge (Typ: T_IPv4AddrArr [▶ 132]).

Beispiel für einen Aufruf in ST:

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.4 F_ScanIPv4AddrIds

Mit der Funktion F_ScanIPv4AddrIds kann ein String mit der (IPv4) Internet-Protocol-Netzwerkadresse in einzelne Adressbytes konvertiert werden. Die einzelnen Adressbytes werden von links nach rechts konvertiert und als Array von Bytes zurückgeliefert. Die Adressbytes haben eine Netzwerk-Byte-Reihenfolge.

FUNCTION F_ScanIPv4AddrIds: T_IPv4AddrArr

Eingänge

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

Name	Typ	Beschreibung
sIPv4	T_IPv4AddrArr	Internet-Protocol-Netzwerkadresse als String (Typ: T_IPv4Addr [▶ 132]). Z. B. 172.16.7.199.

Eingangsparameter	Rückgabeparameter	Beschreibung
sIPv4 ≠ '' (Leerstring) und sIPv4 ≠ '0.0.0.0'	Alle Bytes sind Null	Fehler bei der Konvertierung, überprüfen Sie die Formatierung des sIPv4-Strings.

Beispiel für einen Aufruf in ST:

Im folgenden Beispiel wird ein String mit der Netzwerkadresse 172.16.7.199 in ein Array von Adressbytes konvertiert.

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.5 F_GetCpuCoreIndex

Die Funktion F_GetCpuCoreIndex liefert zu einem Taskindex den Index des CPU-Kerns, auf dem die Task läuft.

Wird als Taskindex 0 übergeben, wird der CPU-Kern-Index der Task ermittelt, in der die Funktion aufgerufen wird. Wird ein ungültiger Taskindex übergeben, liefert die Funktion den CPU-Kern-Index -1 zurück.

Der ermittelte CPU-Kern-Index wird von der Funktion als Rückgabeparameter zurückgeliefert und entspricht dem Wert der Spalte **Core**, die in dem Unterknoten **Echtzeit** unterhalb des SYSTEM-Knotens angezeigt wird.

FUNCTION F_GetCpuCoreIndex: DINT

Eingänge

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

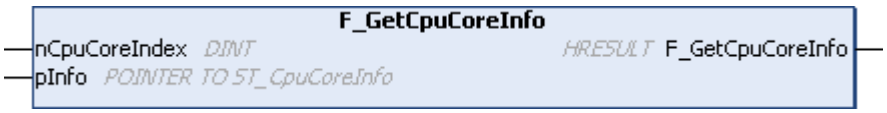
Name	Typ	Beschreibung
nTaskIndex	DINT	Index der Task, deren zugehöriger CPU-Index ermittelt werden soll. Wird als Taskindex 0 übergeben, wird der CPU-Kern-Index der Task ermittelt, in der die Funktion aufgerufen wird.

Siehe auch:

- GETCURTASKINDEX [▶ 19]

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliothe-ken
TwinCAT v3.1.4024.11	PC oder CX (x86, x64, Arm®)	Tc2_System (System) >= 3.4.24.0

4.1.6 F_GetCpuCoreInfo



Die Funktion F_GetCpuCoreInfo liefert Informationen über den CPU-Kern, dessen Index an die Funktion übergeben wird. Die ausgelesenen Informationen enthalten zum Beispiel die Basiszeit und das Core-Limit des angegebenen CPU-Kerns.

Der zu übergebene CPU-Kern-Index kann beispielsweise mit Hilfe der Funktion F_GetCpuCoreIndex [▶ 93] ermittelt werden.

Der CPU-Kern-Index entspricht dem Wert der Spalte **Core**, die in dem Unterknoten **Echtzeit** unterhalb des SYSTEM-Knotens angezeigt wird. Die Informationen, die mit Hilfe der Funktion F_GetCpuCoreInfo über den CPU-Kern ausgelesen werden können, werden ebenfalls in dieser Ansicht dargestellt.

Die Funktion liefert einen Fehlercode als HRESULT zurück (siehe auch: ADS Return Codes [▶ 152]). Dieser zeigt an, ob der Funktionsaufruf erfolgreich war. Wird ein ungültiger CPU-Kern-Index übergeben, liefert die Funktion einen Fehler zurück (0x9811070B = ungültige Parameter-Werte).

FUNCTION F_GetCpuCoreInfo: HRESULT

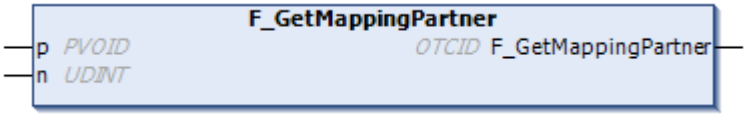
Eingänge

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

Name	Typ	Beschreibung
nCpuCoreIndex	DINT	Index des CPU-Kerns, dessen Informationen ausgelesen werden sollen.
pInfo	POINTER TO ST_CpuCoreInfo	Adresse der Variablen, die die gelesenen Daten aufnehmen soll. Die Adresse muss auf eine Instanz vom Typ ST_CpuCoreInfo [▶ 129] zeigen.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.11	PC oder CX (x86, x64, Arm®)	Tc2_System (System) >= 3.4.24.0

4.1.7 F_GetMappingPartner



Die Funktion F_GetMappingPartner liefert die Objekt-ID (Datentyp: OTCID) der Partnerseite des Mappings.

FUNCTION F_GetMappingPartner: OTCID

Eingänge

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

Name	Typ	Beschreibung
P	PVOID	Speicheradresse der Variablen
n	UDINT	Größe der Variablen in Bytes

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4020	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.8 F_GetMappingStatus

Die Funktion F_GetMappingStatus liefert den aktuellen Mapping- Status einer SPS- Variablen. Die Funktion gibt einen ENUM- Wert (Datentyp: EPlcMappingStatus [▶ 129]) mit den Werten MS_Unmapped, MS_Mapped oder MS_Partial zurück.

FUNCTION F_GetMappingStatus: EPlcMappingStatus

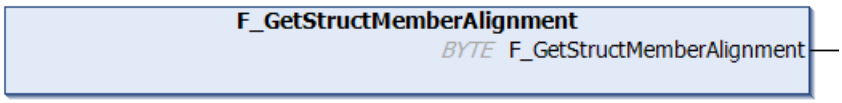
Eingänge

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

Name	Typ	Beschreibung
P	PVOID	Speicheradresse der Variablen
n	UDINT	Größe der Variablen in Bytes

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4020	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.9 F_GetStructMemberAlignment



Die Funktion liefert Informationen über die vom Compiler verwendete Speicherausrichtung (alignment). Die Speicherausrichtung bestimmt die Anordnung der SPS-Datenstrukturelemente im Speicher.

FUNCTION F_GetStructMemberAlignment : BYTE

Eingänge

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

Rückgabeparameter	Beschreibung
1	1 byte alignment (z.B. TwinCAT v2.11, x86-Zielplattform)
2	2 byte alignment
4	4 byte alignment (z.B. TwinCAT v2.11, Arm®-Zielplattform)
8	8 byte alignment

Die folgenden Beispiele zeigen die Anordnung der Datenstrukturelemente im Speicher in Abhängigkeit von der verwendeten Speicherausrichtung.

?? := Füllbyte (padding byte)

Beispiel 1

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

test1 : ST_TEST1;
```

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

Beispiel 2

Durch die Umstellung der Reihenfolge der Strukturelemente ändert sich die Anordnung der Füllbytes. Diese werden jetzt hinten angefügt.

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

test2 : ST_TEST2;
```

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

Beispiel 3

Beim 2-, 4- und 8-Byte-Alignment sind die Elemente ui32 und f64 bereits passend ausgerichtet, so dass keine Füllbytes hinzugefügt werden müssen.

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

test3 : ST_TEST3;
```

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

Beispiel 4

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

TYPE ST_A2
STRUCT
```

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

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

test4 : ST_TEST4;
```

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

Beispiel 5

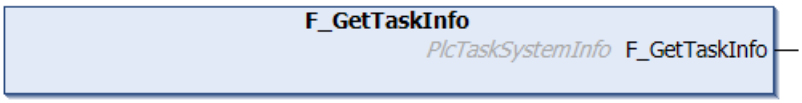
```
TYPE ST_D1
STRUCT
    ui16 : WORD := 16#1234;(* 34 12 *)
    ui8 : BYTE := 16#55;(* 55 *)
END_STRUCT
END_TYPE

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

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

test5 : ST_TEST5;
```

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.10 F_GetTaskInfo



Die Funktion ermittelt die Task System Info der Task, in der sie aufgerufen wird.

FUNCTION F_GetTaskInfo : PlcTaskSystemInfo

Eingänge

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

Rückgabeparameter	Beschreibung
PlcTaskSystemInfo	Task System Info der Task, in der die Funktion aufgerufen wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4026	PC oder CX (x86, x64, Arm®)	Tc2_System (System) >= 3.9.1.0

Sehen Sie dazu auch

GETCURTASKINDEX [▶ 19]

4.1.11 F_RaiseException

Die Funktion ermöglicht es durch Aufruf eine Exception mit einem bestimmten Exception Code zu werfen. Die Funktion sollte nur innerhalb eines __TRY-Blocks verwendet werden, damit das SPS-Programm nicht anhält.

Grundlegende Information zu __TRY, __CATCH, __FINALLY, __ENDTRY

Siehe die Dokumentation zu __TRY, __CATCH, __FINALLY, __ENDTRY, um ein grundlegendes Verständnis vom Exception-Handling zu erlangen.

FUNCTION F_RaiseException

Eingänge

```
VAR_INPUT
    ExceptionCode : UDINT;
END_VAR
```

Name	Typ	Beschreibung
ExceptionCode	UDINT	Angabe des Exception Code. Siehe __SYSTEM.ExceptionCode

Beispiel

```
PROGRAM MAIN
VAR
    nCounter1           : INT;
    nCounter2           : INT;
    nCounter_TRY1       : INT;
    nCounter_TRY2       : INT;
    nCounter_CATCH      : INT;
    aData               : ARRAY[1..cMax] OF INT;
    nIndex              : UINT;

    exc                 : __SYSTEM.ExceptionCode;
    lastExc             : __SYSTEM.ExceptionCode;
END_VAR
VAR CONSTANT
    cMax                : UINT := 10;
END_VAR
```

```
// Counter 1
nCounter1 := nCounter1 + 1;

nIndex := nIndex + 1;

// TRY-CATCH block
__TRY
    nCounter_TRY1 := nCounter_TRY1 + 1;
    IF nIndex > cMax THEN
        F_RaiseException(__SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS);
    END_IF
    nCounter_TRY2 := nCounter_TRY2 + 1;

__CATCH(exc)
    nCounter_CATCH := nCounter_CATCH + 1;

    // Exception logging
    lastExc := exc;

    // Correct the faulty variable values
    IF (exc = __SYSTEM.ExceptionCode.RTSEXCPT_ARRAYBOUNDS) AND (nIndex > cMax) THEN
        nIndex := cMax;
    END_IF

__ENDTRY

aData[nIndex] := 123;

// Counter 2
nCounter2 := nCounter2 + 1;
```

Beispiel mit benutzerspezifischem Exception Code

Eigene benutzerspezifische Exception Codes sollten vor Verwendung mit RTSEXCPT_VENDOR_EXCEPTION_BASE verodert werden.

```
PROGRAM MAIN
VAR
    nCounter1           : INT;
    nCounter2           : INT;
    nCounter_TRY1       : INT;
    nCounter_TRY2       : INT;
    nCounter_CATCH      : INT;
    aData               : ARRAY[1..cMax] OF INT;
    nIndex              : UINT;

    exc                 : __SYSTEM.ExceptionCode;
    lastExc             : __SYSTEM.ExceptionCode;
END_VAR
VAR CONSTANT
    cMyOwnExceptionCode : UDINT := 123;
    cMax                : UINT := 10;
END_VAR
```

```
// Counter 1
nCounter1 := nCounter1 + 1;

nIndex := nIndex + 1;

// TRY-CATCH block
__TRY
    nCounter_TRY1 := nCounter_TRY1 + 1;
    IF nIndex > cMax THEN
        F_RaiseException(__SYSTEM.ExceptionCode.RTSEXCPT_VENDOR_EXCEPTION_BASE OR
cMyOwnExceptionCode);
    END_IF
    nCounter_TRY2 := nCounter_TRY2 + 1;

__CATCH(exc)
    nCounter_CATCH := nCounter_CATCH + 1;

    // Exception logging
    lastExc := exc;

    // Correct the faulty variable values
    IF (exc = __SYSTEM.ExceptionCode.RTSEXCPT_VENDOR_EXCEPTION_BASE OR
cMyOwnExceptionCode) AND (nIndex > cMax) THEN
        nIndex := cMax;
    END_IF

__ENDTRY

aData[nIndex] := 123;

// Counter 2
nCounter2 := nCounter2 + 1;
```

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4026.15	PC oder CX (x86, x64, Arm®)	Tc2_System (System) >= 3.8.1.0

4.1.12 F_SplitPathName

Diese Funktion zerlegt einen vollständigen Pfadnamen in seine vier Einzelkomponenten. Diese werden in den durch sDrive, sDir, sFileName und sExt bezeichneten Strings gespeichert.

FUNCTION F_SplitPathName : BOOL

Eingänge

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

Name	Typ	Beschreibung
sPathName	T_MaxString	Vollständiger Dateiname als String (Typ: T_MaxString [▶ 132]) in der Form: 'X:\DIR\SUBDIR\FILENAME.EXT'.

Ein-/Ausgänge

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

Name	Typ	Beschreibung
sDrive	STRING	Laufwerksbezeichner (Typ: T_MaxString [▶ 132]) mit einem Doppelpunkt ('C:', 'A:' usw.)
sDir	T_MaxString	Verzeichnisname (Typ: T_MaxString [▶ 132]) inklusive dem führenden und abschließenden Backslash ('\BC \INCLUDE\', '\SOURCE\' usw.)
sFileName	T_MaxString	Dateiname (Typ: T_MaxString [▶ 132])
sExt	T_MaxString	Enthält den Punkt und die Namenserweiterung des Dateinamens (Typ: T_MaxString [▶ 132]) ('Beispiel: .C', '.EXE' usw.).

Rückgabeparameter	Beschreibung
TRUE	Kein Fehler
FALSE	Fehler. Überprüfen Sie die Funktionsparameter.

Beispiel für einen Aufruf in ST:

Der Pfadname: C:\TwinCAT\Plc\Project01\Data.txt wird in folgende Einzelkomponenten zerlegt:

sDrive: = 'C:'
sDir: '\TwinCAT\Plc\Project01\'
sFileName: 'Data'
sExt: '.txt'

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.13 SETBIT32

Die Funktion setzt das über eine Bitnummer angegebene Bit in dem ihr übergebenen 32-Bit-Wert und gibt den resultierenden Wert als Ergebnis zurück.

FUNCTION SETBIT32 : DWORD

Eingänge

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

Name	Typ	Beschreibung
inVal32	DWORD	Zu verändernder 32-Bit-Wert.
bitNo	SINT	Nummer des zu setzenden Bits (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet.

Beispiel für den Aufruf der Funktion in FBD:

Hierbei wird das Bit 31 in dem Eingangswert 0 gesetzt. Es ergibt sich der Wert (hex) „80000000“.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.14 CSETBIT32

Die Funktion setzt/rücksetzt das über eine Bitnummer angegebene Bit in dem ihr übergebenen 32-Bit-Wert und gibt den resultierenden Wert als Ergebnis zurück.

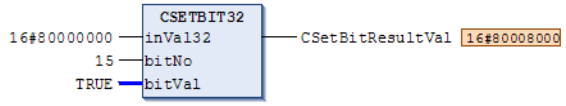
FUNCTION CSETBIT32 : DWORD

Eingänge

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

Name	Typ	Beschreibung
inVal32	DWORD	32-Bit-Wert
bitNo	SINT	Nummer des Bits, das gesetzt bzw. zurückgesetzt werden soll (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet.
bitVal	BOOL	Wert, auf den das Bit gesetzt bzw. zurückgesetzt werden soll (TRUE = 1, FALSE = 0).

Beispiel für den Aufruf der Funktion in FBD:



Hierbei wird das Bit 15 in dem Eingangswert „16#80000000“ auf 1 gesetzt. Das Ergebnis (16#80008000) wird der Variablen CSetBitResultVal zugewiesen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.15 GETBIT32

Die Funktion gibt den Status des über eine Bitnummer angegebenen Bits in dem ihr übergebenen 32-Bit-Wert als boolsches Ergebnis zurück. Der Eingangswert wird nicht verändert.

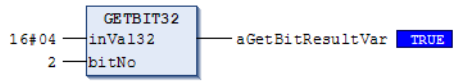
FUNCTION GETBIT32 : BOOL

Eingänge

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

Name	Typ	Beschreibung
inVal32	DWORD	32-Bit-Wert
bitNo	SINT	Nummer des Bits, das gelesen werden soll (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet.

Beispiel für den Aufruf der Funktion in FBD:

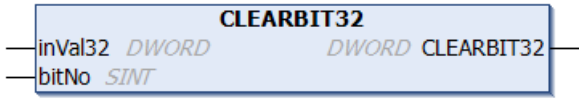


Hierbei wird das Bit 2 in dem Eingangswert „16#04“ abgefragt und der booleschen Variablen aGetBitResultVar zugewiesen. Die Abprüfung ergibt in diesem Beispiel TRUE.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.16 CLEARBIT32



Die Funktion setzt das über eine Bitnummer angegebene Bit in dem ihr übergebenen 32-Bit-Wert auf Null und gibt den resultierenden Wert als Ergebnis zurück.

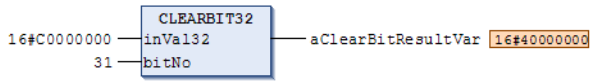
FUNCTION CLEARBIT32 : DWORD

Eingänge

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

Name	Typ	Beschreibung
inVal32	DWORD	Zu verändernder 32-Bit-Wert.
bitNo	SINT	Nummer des zu setzenden Bits (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet.

Beispiel für den Aufruf der Funktion in FBD:

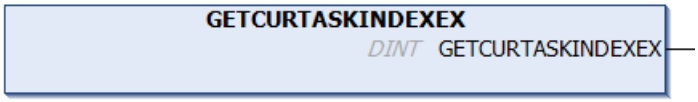


Hierbei wird das Bit 31 in dem Eingangswert „C0000000“ zurückgesetzt. Es ergibt sich der Wert (hex) „40000000“.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.17 GETCURTASKINDEXEX



Die Funktion ermittelt den Taskindex der Task, in der sie aufgerufen wird. Im Gegensatz zum Funktionsbaustein GETCURTASKINDEX [▶ 19] kann unterschieden werden, ob die Funktion im zyklischen Echtzeitkontext aufgerufen wird oder nicht.

FUNCTION GETCURTASKINDEXEX : DINT

Eingänge

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

Rückgabeparameter	Beschreibung
-1	Funktion wird aus dem Windows Context aufgerufen.
0	Funktion wird aus dem Echtzeitkontext, aber nicht von einer zyklischen SPS-Task aufgerufen. Dies ist z. B. bei dem automatischen Aufruf von FB_init-Methoden während der Initialisierung der Fall.
1 bis n	Funktion wird von einer zyklischen SPS-Task aufgerufen. Der Rückgabewert ist der Taskindex.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.18 LPTSIGNAL

Die Funktion setzt ein definiertes Ausgangsbit einer Centronics-Schnittstelle auf logischen High- bzw. Low-Pegel und kann z. B. zur Laufzeitmessung mit Oszilloskopen verwendet werden.

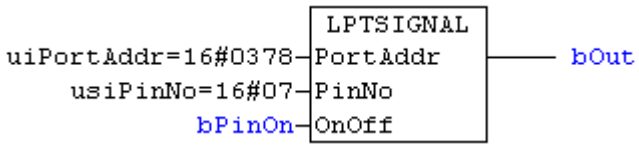
FUNCTION LPTSIGNAL: BOOL

Eingänge

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

Name	Typ	Beschreibung
PortAddr	UINT	Adresse des Ports, welcher für die gewünschte LPT-Schnittstelle zur Verfügung steht.
PinNo	INT	Nummer des Pins (Pin 0 .. 7), der von der PLC beschrieben werden soll.
OnOff	BOOL	Zustand, der an den Pin herausgeschrieben werden soll.

Beispiel für den Aufruf der Funktion in FBD:

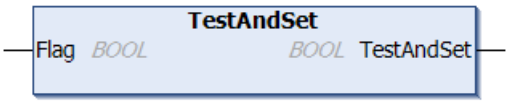


Im Beispiel wird Bit 7 des Ports 378 (hex) auf 1 gesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.1.19 TestAndSet



Mit der Funktion können Sie ein Flag prüfen und setzen, ohne dass dies unterbrochen werden kann. Dadurch können Datenzugriffe synchronisiert werden. Mit TestAndSet kann die Funktionsweise eines Semaphors erreicht werden.

Bei erfolgreichem Funktionsaufruf liefert die Funktion TRUE zurück und auf die gewünschten Daten darf zugegriffen werden. Bei erfolglosem Funktionsaufruf liefert die Funktion FALSE zurück und auf die gewünschten Daten darf nicht zugegriffen werden. In diesem Fall muss applikativ eine Alternativbehandlung vorgesehen werden.

Ein-/Ausgänge

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

Name	Typ	Beschreibung
Flag	BOOL	Boolesches Flag, das geprüft wird • war es FALSE, dann war das Flag frei und wird gesetzt (blockiert von nun an), die Funktion liefert TRUE • war es TRUE, dann war das Flag bereits belegt (blockiert), die Funktion liefert FALSE

Beispiel

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

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

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

    (* ... *)

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

    (* ... *)
END_IF
```

NEGATIV-Beispiel

Vorsicht ist bei einer weiteren Kapselung, z. B. in einem Funktionsbaustein, geboten, da dies die gewünschte atomare Operation zunichtemachen kann. Eine sichere Synchronisierung von Datenzugriffen kann dann nicht mehr erfolgen. Im Folgenden ist ein NEGATIV-Beispiel eingefügt, welches zeigt, wie die Funktion NICHT verwendet werden darf. Sollten bei dieser Implementierung zwei Kontexte zugleich Zugriff erbitten, so könnten beide davon ausgehen, dass der Zugriff erlaubt ist und es würde ein zeitgleicher, ungesicherter Zugriff auf die Daten stattfinden.

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

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

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

    (* ... *)

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

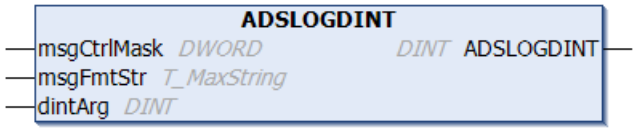
Mit dem Funktionsbaustein FB_IecCriticalSection [▶ 14] wird die Verwendung von kritischen Bereichen als alternatives Mutex-Verfahren angeboten.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.2 ADS-Funktionen

4.2.1 ADSLOGDINT



Die Funktion gibt bei Aufruf eine Message-Box mit einem vorgebbaren Text auf den Bildschirm aus und schreibt einen Eintrag in das Ereignislogbuch des Systems. Da ein SPS-Programm zyklisch abgearbeitet wird, muss z. B. eine Message-Box flankengesteuert ausgegeben werden. Am einfachsten lässt sich dies mit einem vorgeschalteten R_TRIG- oder F_TRIG-Funktionsbaustein erreichen (siehe auch Beispiel im weiteren Verlauf).

Bei der ADSLOGDINT-Funktion kann in den auszugebenden Text ein DINT-Wert (4 Byte vorzeichenbehafteter Integer) an eine von dem Benutzer vorgebbare Stelle eingearbeitet werden. Dazu muss der angelegte Formatstring an der gewünschten Stelle die Zeichenfolge %d enthalten. Der Rückgabeparameter enthält den Funktionsfehlercode oder 0 falls erfolgreich.

FUNCTION ADSLOGDINT : DINT

Eingänge

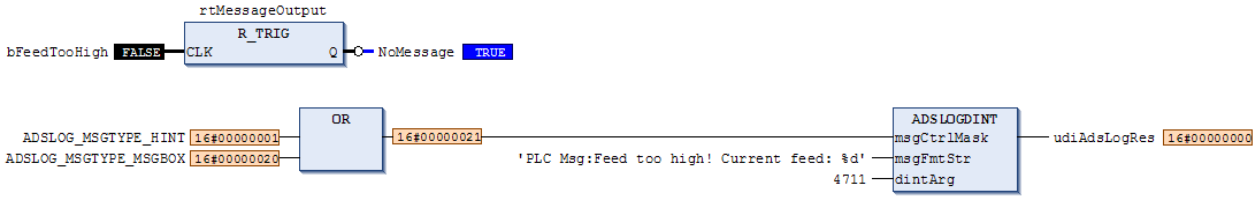
```
VAR_INPUT
    msgCtrlMask : DWORD;
    msgFmtStr   : T_MaxString;
    dintArg     : DINT;
END_VAR
```

Name	Typ	Beschreibung
msgCtrlMask	DWORD	Kontrollmaske, die den Typ und die Wirkung der Meldungsausgabe bestimmt (siehe separate Tabelle).
msgFmtStr	T_MaxString [▶ 132]	Enthält die auszugebende Meldung. Sie kann das Formatierungszeichen %d für die Ausgabe eines DINT-Wertes an beliebiger Stelle enthalten. **Information:** Die Länge der Meldung ist auf 253 Bytes begrenzt (entspricht 253 Zeichen bei Strings im Standardformat).
dintArg	DINT	Enthält den in die Meldung einzufügenden numerischen Wert.

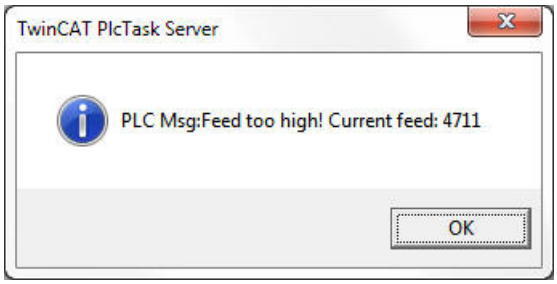
Konstante	Beschreibung
ADSLOG_MSGTYPE_HINT	Meldungstyp ist Hinweis.
ADSLOG_MSGTYPE_WARN	Meldungstyp ist Warnung.
ADSLOG_MSGTYPE_ERROR	Meldungstyp ist Fehler.
ADSLOG_MSGTYPE_LOG	Meldung wird in das Logbuch geschrieben.
ADSLOG_MSGTYPE_MSGBOX	Meldung wird in einer Messagebox ausgegeben. **Information:** Diese Funktionalität ist nicht für Windows CE verfügbar.
ADSLOG_MSGTYPE_STRING	Meldung ist direkt angegebener String (default).

Die Kontrollmasken können in der gewünschten Kombination mit ODER verknüpft werden.

Beispiel für den Aufruf der Funktion in FBD:



Die resultierende Message-Box:



Hierbei wird der DINT-Wert 4711 in eine Meldung eingefügt. Die Einfügestelle ist durch das Zeichen %d im Formatstring markiert.

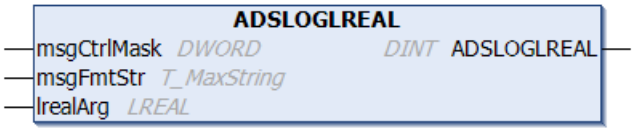
Beispiel für den Aufruf der Funktion in ST:

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.2.2 ADSLOGLREAL



Die Funktion gibt bei Aufruf eine Message-Box mit einem vorgebbaren Text auf den Bildschirm aus und schreibt einen Eintrag in das Ereignislogbuch des Systems. In den auszugebenden Text kann ein LREAL-Wert (Gleitkommazahl) an eine von dem Benutzer vorgebbare Stelle eingearbeitet werden. Dazu muss der angelegte Formatstring an der gewünschten Stelle die Zeichenfolge ‚%f' enthalten. Bedenken Sie, dass die Funktion flankengesteuert aufgerufen werden muss, wie im Beispiel gezeigt (siehe auch Hinweis bei Beschreibung ADSLOGDINT [▶ 107]). Der Rückgabeparameter enthält den Funktionsfehlercode oder 0 falls erfolgreich.

FUNCTION ADSLOGLREAL : DINT

Eingänge

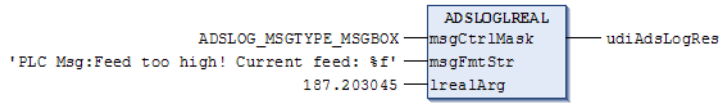
```
VAR_INPUT
    msgCtrlMask : DWORD;
    msgFmtStr   : T_MaxString;
    lrealArg    : LREAL;
END_VAR
```

Name	Typ	Beschreibung
msgCtrlMask	DWORD	Kontrollmaske, die den Typ und die Wirkung der Meldungsausgabe bestimmt (siehe separate Tabelle).
msgFmtStr	T_MaxString [▶ 132]	Enthält die auszugebende Meldung. Sie kann das Formatierungszeichen %f für die Ausgabe eines LREAL-Wertes an beliebiger Stelle enthalten. **Information:** Die Länge der Meldung ist auf 253 Bytes begrenzt (entspricht 253 Zeichen bei Strings im Standardformat).
lrealArg	LREAL	Enthält den in die Meldung einzufügenden numerischen Wert.

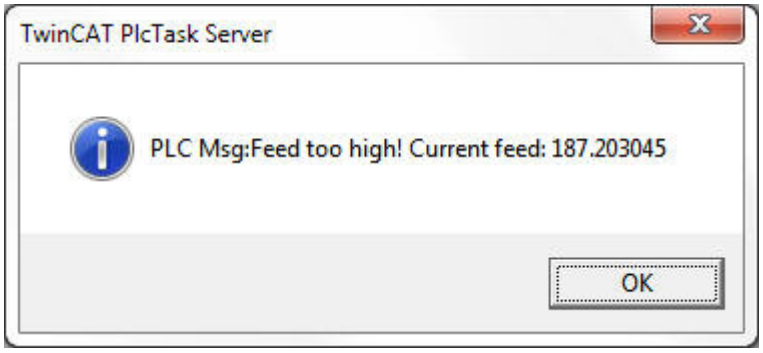
Konstante	Beschreibung
ADSLOG_MSGTYPE_HINT	Meldungstyp ist Hinweis.
ADSLOG_MSGTYPE_WARN	Meldungstyp ist Warnung.
ADSLOG_MSGTYPE_ERROR	Meldungstyp ist Fehler.
ADSLOG_MSGTYPE_LOG	Meldung wird in das Logbuch geschrieben.
ADSLOG_MSGTYPE_MSGBOX	Meldung wird in einer Messagebox ausgegeben. **Information:** Diese Funktionalität ist nicht für Windows CE verfügbar.
ADSLOG_MSGTYPE_STRING	Meldung ist direkt angegebener String (default).

Die Kontrollmasken können in der gewünschten Kombination mit ODER verknüpft werden.

Beispiel für den Aufruf der Funktion in FBD:



Die resultierende Message-Box:



Hierbei wird der LREAL-Wert 187.203045 in eine Meldung eingefügt. Die Einfügestelle ist durch das Zeichen %f im Formatstring markiert. Die Zahl wird bei der Ausgabe nach der sechsten Nachkommastelle abgeschnitten.

Beispiel für den Aufruf der Funktion in ST:

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

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

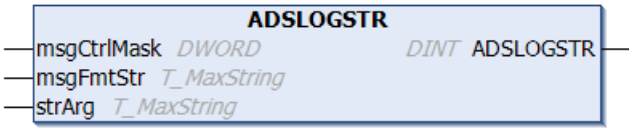
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

Sehen Sie dazu auch

- ADSLOGDINT [▶ 107]

4.2.3 ADSLOGSTR



Die Funktion gibt bei Aufruf eine Message-Box mit einem vorgebbaren Text auf den Bildschirm aus und schreibt einen Eintrag in das Ereignislogbuch des Systems. In den auszugebenden Text kann ein STRING (Zeichenkette) an eine von dem Benutzer vorgebbare Stelle eingearbeitet werden. Dazu muss der angelegte Formatstring an der gewünschten Stelle die Zeichenfolge %s enthalten. Bedenken Sie, dass die Funktion flankengesteuert aufgerufen werden muss, wie im Beispiel gezeigt (siehe auch Hinweis bei Beschreibung ADSLOGDINT [▶ 107]). Der Rückgabeparameter enthält den Funktionsfehlercode oder 0 falls erfolgreich.

FUNCTION ADSLOGSTR : DINT

Eingänge

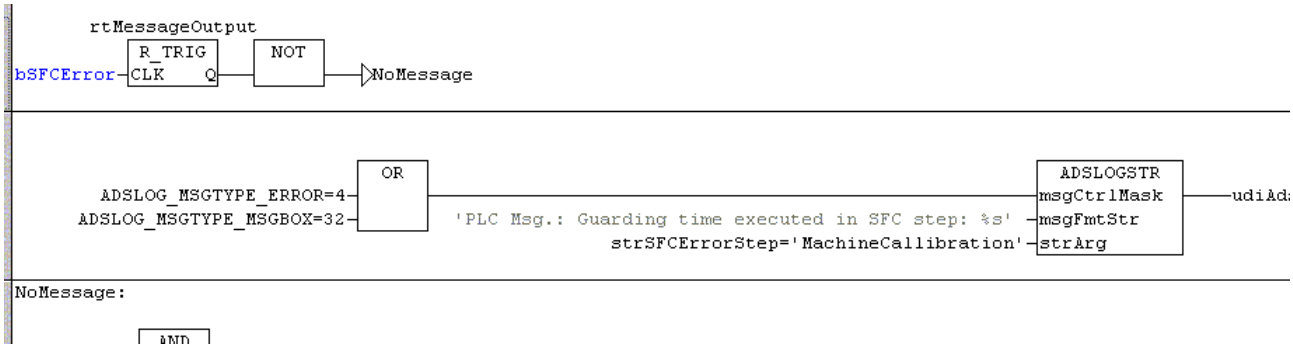
```
VAR_INPUT
    msgCtrlMask   : DWORD;
    msgFmtStr     : T_MaxString;
    strArg        : T_MaxString;
END_VAR
```

Name	Typ	Beschreibung
msgCtrlMask	DWORD	Kontrollmaske, die den Typ und die Wirkung der Meldungsausgabe bestimmt (siehe separate Tabelle).
msgFmtStr	T_MaxString [▶ 132]	Enthält die auszugebende Meldung. Sie kann das Formatierungszeichen %s für die Ausgabe eines Text-Argumentes an beliebiger Stelle enthalten. **Information:** Die Länge der Meldung ist auf 253 Bytes begrenzt (entspricht 253 Zeichen bei Strings im Standardformat).
strArg	T_MaxString [▶ 132]	Enthält den in die Meldung einzufügenden String.

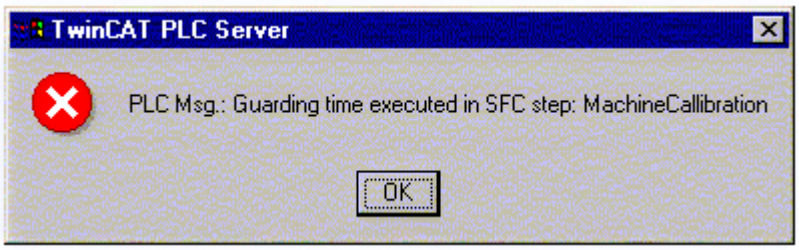
Konstante	Beschreibung
ADSLOG_MSGTYPE_HINT	Meldungstyp ist Hinweis.
ADSLOG_MSGTYPE_WARN	Meldungstyp ist Warnung.
ADSLOG_MSGTYPE_ERROR	Meldungstyp ist Fehler.
ADSLOG_MSGTYPE_LOG	Meldung wird in das Logbuch geschrieben.
ADSLOG_MSGTYPE_MSGBOX	Meldung wird in einer Messagebox ausgegeben. **Information:** Diese Funktionalität ist nicht für Windows CE verfügbar.
ADSLOG_MSGTYPE_STRING	Meldung ist direkt angegebener String (default).

Die Kontrollmasken können in der gewünschten Kombination mit ODER verknüpft werden.

Beispiel für den Aufruf der Funktion in FBD:



Die resultierende Message-Box:



Hierbei wird der String, welcher in der Variable strSFCErrorStep steht, von dem SPS-Programmierer in die Meldung eingefügt. Die Einfügestelle ist durch das Zeichen %s im Formatstring markiert.

Beispiel für den Aufruf der Funktion in ST:

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

Sehen Sie dazu auch

ADSLOGDINT [▶ 107]

4.2.4 F_CreateAmsNetId

Die Funktion generiert einen formatierten NetId-String (Typ: T_AmsNetId [▶ 130]) und liefert diesen als Rückgabeparameter zurück (z. B. '127.16.17.3.1.1').

FUNCTION F_CreateAmsNetId : T_AmsNetId

Eingänge

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

Name	Typ	Beschreibung
nIds	T_AmsNetIdArr	Byte-Array (Typ: T_AmsNetIdArr [▶ 130]). Jedes Byte entspricht einer Nummer der Netzwerkadresse. Die Adressbytes haben eine Netzwerk-Byte-Reihenfolge.

Beispiel für einen Aufruf in ST:

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.2.5 F_ScanAmsNetIds

Mit der Funktion F_ScanAmsNetIds kann ein String mit der TwinCAT-Netzwerkadresse in einzelne Adressbytes konvertiert werden. Die einzelnen Adressbytes werden von links nach rechts konvertiert und als Array von Bytes zurückgeliefert (Typ: T_AmsNetIdArr [▶ 130]). Die Adressbytes haben eine Netzwerk-Byte-Reihenfolge.

FUNCTION F_ScanAmsNetIds : T_AmsNetIdArr

Eingänge

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

Name	Typ	Beschreibung
sNetID	T_AmsNetID	TwinCAT-Netzwerkadresse als String (Typ: T_AmsNetId [▶ 130]). Z. B.: '127.16.17.3.1.1'

Eingangsparameter	Rückgabeparameter	Beschreibung
sNetID ≠ '' (Leerstring) und sNetID ≠ '0.0.0.0.0.0'	Alle Bytes sind Null	Fehler bei der Konvertierung, überprüfen Sie die Formatierung des sNetID-Strings.

Beispiel für einen Aufruf in ST:

Im folgenden Beispiel wird ein String mit der Netzwerkadresse '127.16.17.3.1.1' in ein Array von Adressbytes konvertiert.

```
PROGRAM MAIN
VAR
    ids    : T_AmsNetIDArr;
    sNetID : T_AmsNetID := '127.16.17.3.1.1';
END_VAR

ids := F_ScanAmsNetIds( sNetID );
(* Result: ids[0]:=127, ids[1]:=16, ids[2]:=17, ids[3]:=3, ids[4]:=1, ids[5]:=1 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.3 Character-Funktionen

4.3.1 F_ToCHR

Die Funktion konvertiert den ASCII-Code in ein String-Zeichen.

FUNCTION F_ToCHR: STRING

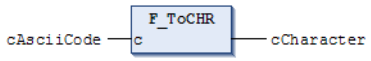
Eingänge

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

Name	Typ	Beschreibung
c	BYTE	Zu konvertierender ASCII-Code

Beispiel für den Aufruf der Funktion in FBD:

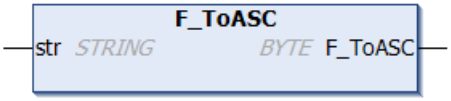
```
PROGRAM P_TEST
VAR
    sCharacter : STRING(1) := '';
    cAsciiCode : BYTE := 16#31;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.3.2 F_ToASC



Die Funktion konvertiert ein String-Zeichen in den ASCII-Code. Es wird nur das erste Zeichen des Strings konvertiert. Ein Leerstring liefert eine Null zurück.

FUNCTION F_ToASC : BYTE

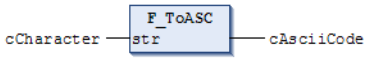
Eingänge

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

Name	Typ	Beschreibung
str	STRING	Zu konvertierender String

Beispiel für den Aufruf der Funktion in FBD:

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

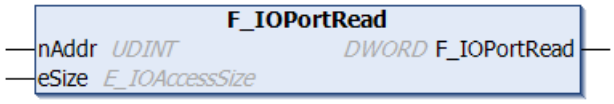


Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.4 I/O-Portzugriff

4.4.1 F_IOPortRead



Ein digitaler I/O-Port ist in den meisten Fällen eine ein Byte breite I/O-Position, die entweder in den Speicher oder als Port abgebildet wird. Wenn Sie einen Wert an diese Stelle schreiben, wird das elektrische Signal an den Ausgabe-Pins entsprechend den geschriebenen Bits geändert. Wenn Sie einen Wert aus der Eingabe-Position auslesen, wird das aktuelle logische Niveau an den Eingabe-Pins als individueller Bit-Wert zurückgegeben.

Mit der Funktion F_IOPortRead kann eine eSize-breite I/O-Position ausgelesen werden. Der gelesene Wert wird von der Funktion als Rückgabeparameter zurückgeliefert. Siehe auch die Beschreibung der F_IOPortWrite [▶ 115] Funktion.

FUNCTION F_IOPortRead : DWORD

Eingänge

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

Name	Typ	Beschreibung
nAddr	UDINT	I/O-Portadresse
eSize	E_IOAccessSize	Anzahl der zu lesenden Datenbytes (Typ: E_IOAccessSize [▶ 126])

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.4.2 F_IOPortWrite

Mit der Funktion F_IOPortWrite kann eine eSize-breite I/O-Position beschrieben werden. Siehe auch die Beschreibung der F_IOPortRead [▶ 115] Funktion.

HINWEIS

Beschädigung der Hardware

Ein direkter Hardwarezugriff stellt kein Problem dar, solange Daten nur gelesen werden. Ein schreibender Zugriff kann zu Abstürzen führen und/oder die Hardware/Daten auf den Speichermedien zerstören. Mit dieser Funktion kann die Hardware so beschädigt werden, dass sie nicht mehr bootfähig ist.

FUNCTION F_IOPortWrite : BOOL

Eingänge

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

Name	Typ	Beschreibung
nAddr	UDINT	I/O-Portadresse
eSize	E_IOAccessSize	Anzahl der Datenbytes die geschrieben werden sollen (Typ: E_IOAccessSize [▶ 126]).
nValue	DWORD	Wert, der geschrieben werden soll.

Rückgabeparameter	Beschreibung
TRUE	Kein Fehler
FALSE	Fehler

Beispiel in ST:

Im folgenden Beispiel wurde mit Hilfe der I/O-Port Funktionen ein SPS-Baustein zur direkten Ansteuerung des PC-Lautsprechers implementiert.

Interface:

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

Implementation:

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

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

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

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

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

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

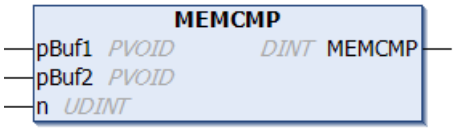
```
        nState := 1;
    ELSE
        nState := 100;
    END_IF
END_IF

1:
timer( );
IF timer.Q THEN

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

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

END_CASE

Test application:

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


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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.5 Memory-Funktionen

4.5.1 MEMCMP



Mit der Funktion MEMCMP können die Werte der SPS-Variablen in zwei unterschiedlichen Speicherbereichen verglichen werden.

FUNCTION MEMCMP : DINT

Eingänge

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

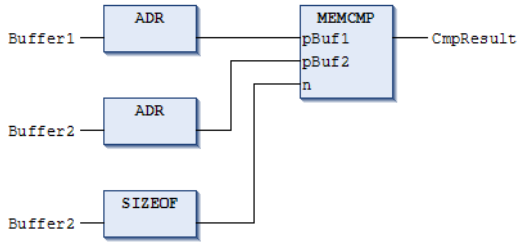
Name	Typ	Beschreibung
pBuf1	PVOID	Anfangsadresse des ersten Speicherbereichs (der erste Datenpuffer).
pBuf2	PVOID	Anfangsadresse des zweiten Speicherbereichs (der zweite Datenpuffer).
n	UDINT	Anzahl der zu vergleichenden Bytes.

Die Funktion vergleicht die ersten n-Bytes in den beiden Datenpuffern und liefert einen Wert, der deren Verhältnis entspricht.

Rückgabeparameter	Verhältnis des ersten unterschiedlichen Bytes im ersten und zweiten Datenpuffer
-1	pBuf1 kleiner als pBuf2
0	pBuf1 identisch mit pBuf2
1	pBuf1 größer als pBuf2
0xFF	Falsche Parameterwerte. pBuff1 = 0 oder pBuff2 = 0 oder n = 0

Beispiel für einen Aufruf in FUP

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

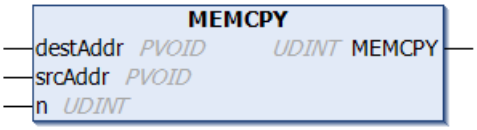


Im Beispiel werden 4 Byte Daten vom Buffer2 mit dem Buffer1 verglichen. Das erste unterschiedliche Datenbyte ist im Buffer1 größer als in Buffer2.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.5.2 MEMCPY



Mit der Funktion MEMCPY können Werte der SPS-Variablen von einem Speicherbereich in einen anderen kopiert werden.

HINWEIS

Systemabsturz oder Zugriff auf unerlaubte Speicherbereiche

Da mit der Funktion direkt auf den physikalischen Speicher zugegriffen wird, ist bei deren Anwendung besondere Vorsicht geboten. Falsche Parameterwerte können zu einem Systemabsturz oder einem Zugriff auf unerlaubte Speicherbereiche führen.

Das Verhalten von MEMCPY ist undefiniert, wenn sich der Ziel- und der Quell-Speicherbereich überschneiden. Das ist z. B. dann der Fall, wenn mehrere in einem Array gespeicherte Werte eine Position nach vorne oder nach hinten verschoben werden sollen. Nutzen Sie in einem solchen Fall die Funktion MEMMOVE [▶ 119].

FUNCTION MEMCPY : UDINT

Eingänge

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

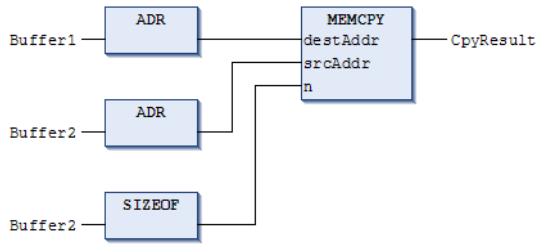
Name	Typ	Beschreibung
destAddr	PVOID	Anfangsadresse des Ziel-Speicherbereichs.
srcAddr	PVOID	Anfangsadresse des Quell-Speicherbereichs.
n	UDINT	Anzahl der zu kopierenden Bytes.

Die Funktion kopiert n-Bytes ab dem Speicherbereich mit der Anfangsadresse srcAddr in den Speicherbereich mit der Anfangsadresse destAddr.

Rückgabeparameter	Bedeutung
0	Falsche Parameterwerte. destAddr == 0 oder srcAddr == 0 oder n == 0
> 0	Bei Erfolg, die Anzahl der kopierten Bytes (n).

Beispiel für einen Aufruf in FUP

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



Im Beispiel werden 4 Byte vom Buffer2 nach Buffer1 kopiert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.5.3 MEMMOVE

Die Funktion MEMMOVE verwenden Sie, um Werte von einem Speicherbereich in einen anderen zu kopieren, die Speicherbereiche dürfen sich dabei überlappen.

HINWEIS

Systemabsturz oder Zugriff auf unerlaubte Speicherbereiche

Weil mit der Funktion direkt auf den physikalischen Speicher zugegriffen wird, ist bei deren Anwendung besondere Vorsicht geboten. Falsche Parameterwerte können zu einem Systemabsturz oder einem Zugriff auf unerlaubte Speicherbereiche führen.

FUNCTION MEMMOVE : UDINT

Eingänge

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

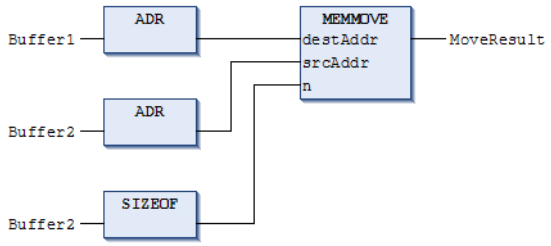
Name	Typ	Beschreibung
destAddr	PVOID	Anfangsadresse des Ziel-Speicherbereichs.
srcAddr	PVOID	Anfangsadresse des Quell-Speicherbereichs.
n	UDINT	Anzahl der zu kopierenden Bytes.

Die Funktion kopiert n-Bytes ab dem Speicherbereich mit der Anfangsadresse srcAddr in den Speicherbereich mit der Anfangsadresse destAddr.

Rückgabeparameter	Bedeutung
0	Falsche Parameterwerte. destAddr == 0 oder srcAddr == 0 oder n == 0
> 0	Bei Erfolg, die Anzahl der kopierten Bytes (n).

Beispiel für einen Aufruf in FUP:

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

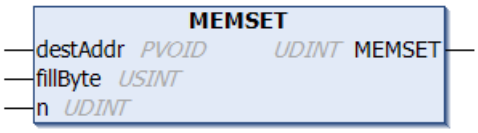


Im Beispiel werden 4 Byte vom Buffer2 nach Buffer1 verschoben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.5.4 MEMSET



Mit der Funktion MEMSET können SPS-Variablen in einem Speicherbereich auf einen bestimmten Wert gesetzt werden.

HINWEIS

Systemabsturz oder Zugriff auf unerlaubte Speicherbereiche

Weil mit der Funktion direkt auf den physikalischen Speicher zugegriffen wird, ist bei deren Anwendung besondere Vorsicht geboten. Falsche Parameterwerte können zu einem Systemabsturz oder einem Zugriff auf unerlaubte Speicherbereiche führen.

FUNCTION MEMSET : UDINT

Eingänge

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

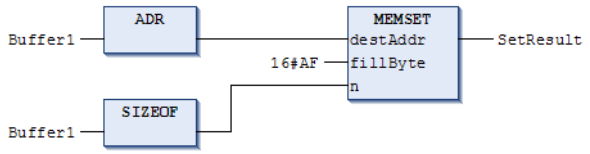
Name	Typ	Beschreibung
destAddr	PVOID	Anfangsadresse des zu setzenden Speicherbereichs.
fillByte	USINT	Wert der Füll-Bytes.
n	UDINT	Anzahl der zu setzenden Bytes.

Die Funktion füllt n-Bytes ab dem Speicherbereich mit der Anfangsadresse destAddr mit den Werten fillByte.

Rückgabeparameter	Bedeutung
0	Falsche Parameterwerte. destAddr == 0 oder n == 0
> 0	Bei Erfolg, die Anzahl der gesetzten Bytes (n).

Beispiel für einen Aufruf in FUP

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



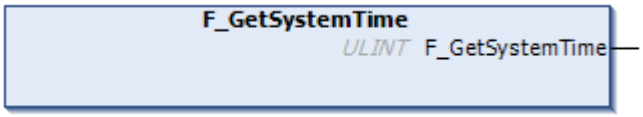
Im Beispiel werden 4 Byte im Buffer1 auf den Wert 0xAF gesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.6 Zeit-Funktionen

4.6.1 F_GetSystemTime



Mit dieser Funktion kann der Betriebssystem-Zeitstempel ausgelesen werden. Der Zeitstempel ist ein 64-Bit-Integer-Wert, mit einer Genauigkeit von 100 ns, welcher bei jedem Aufruf der SPS aktualisiert wird. Er kann unter anderem für Timing-Aufgaben oder Zeitmessungen eingesetzt werden. Eine Einheit entspricht 100 ns. Die Zeit repräsentiert die Anzahl der 100 ns-Intervalle seit dem 1. Januar 1601 (UTC).

Ausgänge

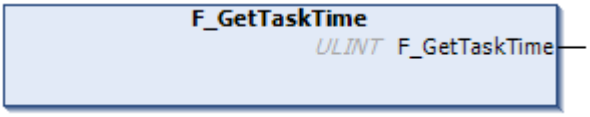
```
VAR_OUTPUT
    F_GetSystemTime   : ULINT;
END_VAR
```

Der Rückgabewert enthält den Zeitstempel.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.17.0

4.6.2 F_GetTaskTime



Mit dieser Funktion kann die Startzeit der Task (Zeitpunkt, zu dem die Task starten sollte) ausgelesen werden. Die Funktion liefert immer die Startzeit der Task, in der die Funktion aufgerufen wurde. Der Zeitstempel ist ein 64-Bit-Integer-Wert, mit einer Genauigkeit von 100ns und kann unter anderem für Timing-Aufgaben oder Zeitmessungen eingesetzt werden. Eine Einheit entspricht 100ns. Die Zeit repräsentiert die Anzahl der 100ns-Intervalle seit dem 1. Januar 1601.

Ausgänge

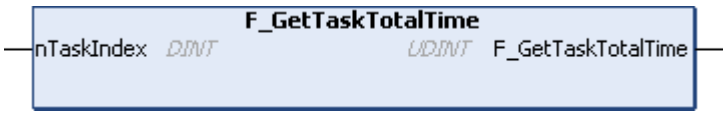
```
VAR_OUTPUT
    F_GetTaskTime : ULINT;
END_VAR
```

Der Rückgabewert enthält den Zeitstempel.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.17.0

4.6.3 F_GetTaskTotalTime



Die Funktion F_GetTaskTotalTime liefert zu einem Taskindex die Gesamtausführungszeit dieser Task aus dem letzten Zyklus. Die Gesamtausführungszeit entspricht der Summe der Rechenzeiten aller Module, die bei der Task angemeldet sind.

Wird als Taskindex 0 übergeben, wird der Wert für die Task ermittelt, in der die Funktion aufgerufen wird. Wird ein ungültiger Taskindex übergeben, liefert die Funktion die Gesamtausführungszeit 0 zurück.

Die ermittelte Gesamtausführungszeit wird als Vielfaches von 100ns angegeben und wird von der Funktion als Rückgabeparameter zurückgeliefert.

FUNCTION F_GetTaskTotalTime: UDINT

Eingänge

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

Name	Typ	Beschreibung
nTaskIndex	DINT	Index der Task, deren Gesamtausführungszeit ermittelt werden soll. Wird als Taskindex 0 übergeben, wird der Wert für die Task ermittelt, in der die Funktion aufgerufen wird.

Siehe auch:

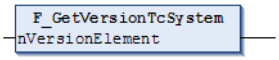
- GETCURTASKINDEX [▶ 19]

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.11	PC oder CX (x86, x64, Arm®)	Tc2_System (System) >= 3.4.24.0

4.7 [veraltet]

4.7.1 F_GetVersionTcSystem

Diese Funktion ist veraltet und sollte nicht verwendet werden. Verwenden Sie stattdessen die globale Konstante stLibVersion_Tc2_System [▶ 140], um Versionsinformationen der SPS-Bibliothek auszulesen.



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

FUNCTION F_GetVersionTcSystem : UINT

Eingänge

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

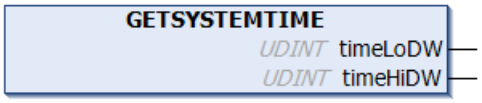
Name	Typ	Beschreibung
nVersionElement	INT	**nVersionElement** : Versionselement, das gelesen werden soll. Mögliche Parameter: • 1 : major number; • 2 : minor number; • 3 : revision number;

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.7.2 GETSYSTEMTIME

Dieser Funktionsbaustein wird durch die neuere Funktion F_GetSystemTime() abgelöst, welche mit einem anstatt zwei Rückgabewerten auskommt.



Mit diesem Baustein kann der Betriebssystem-Zeitstempel ausgelesen werden. Der Zeitstempel ist ein 64-Bit-Integer-Wert, mit einer Genauigkeit von 100ns, welcher bei jedem Aufruf der SPS aktualisiert wird. Er kann unter anderem für Timing-Aufgaben oder Zeitmessungen eingesetzt werden. Eine Einheit entspricht 100ns. Die Zeit repräsentiert die Anzahl der 100ns-Intervalle seit dem 1. Januar 1601.

Siehe: F_GetSystemTime [▶ 122]

Eingänge

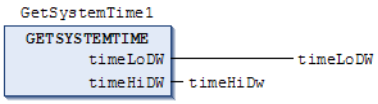
```
VAR_INPUT
(*none*)
END_VAR
```

Ausgänge

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

Name	Type	Beschreibung
timeLoDW	UDINT	Enthält die niederwertigeren 4 Byte des Zeitstempels.
timeHiDW	UDINT	Enthält die höherwertigeren 4 Byte des Zeitstempels.

Beispiel für den Aufruf des Bausteins in FBD:



Das Beispiel zeigt den Aufruf des Bausteins über die Instanz GetSystemTime1 und liefert den 64-bit, ganzzahligen Wert (hex), 1BCD6EAB05C4E60 als Zeitstempel.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

4.7.3 GETTASKTIME

Dieser Funktionsbaustein wird durch die neuere Funktion F_GetTaskTime() abgelöst, welche mit einem anstatt zwei Rückgabewerten auskommt.

Mit diesem Baustein kann die Startzeit der Task (Zeitpunkt, zu dem die Task starten sollte) ausgelesen werden. Der Funktionsbaustein liefert immer die Startzeit der Task, in der die Bausteininstanz aufgerufen wurde. Der Zeitstempel ist ein 64-Bit-Integer-Wert, mit einer Genauigkeit von 100ns und kann unter anderem für Timing-Aufgaben oder Zeitmessungen eingesetzt werden. Eine Einheit entspricht 100ns. Die Zeit repräsentiert die Anzahl der 100ns-Intervalle seit dem 1. Januar 1601.

Siehe: F_GetTaskTime [▶ 122]

Eingänge

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

Ausgänge

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

Name	Type	Beschreibung
timeLoDW	UDINT	Enthält die niederwertigeren 4 Byte des Zeitstempels.
timeHiDW	UDINT	Enthält die höherwertigeren 4 Byte des Zeitstempels.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5 Datentypen

5.1 E_IOAccessSize

Bytegröße der I/O-Position (Anzahl der Bytes, die gelesen oder geschrieben werden sollen).

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.2 E_OpenPath

Über eine Variable von diesem Typ kann ein generischer oder ein TwinCAT-Systempfad auf dem Zielgerät zum Öffnen einer Datei ausgewählt werden.

```
TYPE E_OpenPath :
(
    PATH_GENERIC :=1, (* search/open/create files in selected/generic folder *)
    PATH_BOOTPRJ,     (* search/open/create files in the TwinCAT/
Boot directory (adds the extension .wbp) *)
    PATH_BOOTDATA,    (* reserved for future use*)
    PATH_BOOTPATH,    (* refers to the TwinCAT/Boot directory without adding an extension (.wbp) *)
    PATH_USERPATH1 :=11, (*reserved for future use*)
    PATH_USERPATH2,   (*reserved for future use*)
    PATH_USERPATH3,   (*reserved for future use*)
    PATH_USERPATH4,   (*reserved for future use*)
    PATH_USERPATH5,   (*reserved for future use*)
    PATH_USERPATH6,   (*reserved for future use*)
    PATH_USERPATH7,   (*reserved for future use*)
    PATH_USERPATH8,   (*reserved for future use*)
    PATH_USERPATH9    (*reserved for future use*)
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.3 E_SeekOrigin

Eine Variable von diesem Typ bestimmt den Ursprungspunkt beim Bewegen des Dateizeigers.

```
TYPE E_SeekOrigin :
(
    SEEK_SET   := 0, (* Seek from beginning of file *)
    SEEK_CUR,        (* Seek from current position of file pointer *)
    SEEK_END         (* Seek from the end of file *)
);
END_TYPE
```

Wert	Beschreibung
SEEK_SET	Bewegt den Dateizeiger relativ zum Dateianfang
SEEK_CUR	Bewegt den Dateizeiger relativ zur aktuellen Zeigerposition
SEEK_END	Bewegt den Dateizeiger relativ zum Dateiende

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.4 E_TcEventClass

```
TYPE E_TcEventClass :
(
    TCEVENTCLASS_NONE        :=0, (* No class *)
    TCEVENTCLASS_MAINTENANCE :=1, (* Maintenance hint *)
    TCEVENTCLASS_MESSAGE     :=2, (* Message *)
    TCEVENTCLASS_HINT        :=3, (* Hint *)
    TCEVENTCLASS_STATEINFO   :=4, (* State information *)
    TCEVENTCLASS_INSTRUCTION :=5, (* Instruction *)
    TCEVENTCLASS_WARNING     :=6, (* Warning *)
    TCEVENTCLASS_ALARM       :=7, (* Alarm *)
    TCEVENTCLASS_PARAMERROR  :=8 (* Parameter error *)
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.5 E_TcEventClearModes

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.6 E_TcEventPriority

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.7 E_TcEventStreamType

```
TYPE E_TcEventStreamType :
(
    TCEVENTSTREAM_INVALID       := 0, (* no source name, no prog id *)
    TCEVENTSTREAM_SIMPLE,             (* no source name, no prog id *)
    TCEVENTSTREAM_NORMAL,             (* source name AND prog id *)
    TCEVENTSTREAM_NOSOURCE,           (* no source name, but prog id *)
    TCEVENTSTREAM_CLASSID,            (* source name AND class id *)
    TCEVENTSTREAM_CLSNOSRC,           (* no source name but class id *)
    TCEVENTSTREAM_READCLASSCOUNT,     (* *)
    TCEVENTSTREAM_MAXTYPE             (* no source name but class id *)
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.8 E_TcMemoryArea

Die Funktion F_CheckMemoryArea [▶ 90] liefert Information darüber, in welchem Speicherbereich sich die angefragte Variable mit angegebener Größe befindet. Hierzu wird ein Rückgabewert vom Typ E_TcMemoryArea verwendet.

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

Name	Beschreibung
Unknown	Der Speicherbereich ist unbekannt. Hierbei kann es sich beispielsweise um Speicher im Windows-Kontext handeln. Der Speicherbereich wird ebenfalls als unbekannt ausgegeben, falls die angegebene Speichergröße dazu führt, dass zwei unterschiedliche Speicherbereiche beteiligt sind. Des Weiteren wird der Speicherbereich als unbekannt ausgegeben, wenn es sich um Stack-Speicher handelt.
Static	Es handelt sich um statischen SPS-Speicher.
Dynamic	Es handelt sich um dynamischen allokierten Speicher, welcher während der Laufzeit oder auch während der Initialisierungsphase der SPS allokiert wurde.
CNC	Es handelt sich um Speicher des CNC-Treibers.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.4022	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.9 E_UsrLED_Color

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

5.10 EPlcMappingStatus

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

Dieser Datentyp ist im globalen Typsystem definiert.

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4020	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.11 ST_AmsAddr

Eine Variable von diesem Typ enthält die TwinCAT-Netzwerkadresse.

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

Name	Typ	Beschreibung
netId	T_AmsNetIdArr	AMS-Netzwerkadresse (Typ: T_AmsNetIdArr [▶ 130])
port	T_AmsPort	AMS-Portnummer (Typ: T_AmsPort [▶ 130])

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.12 ST_CpuCoreInfo

Eine Variable von diesem Typ enthält Informationen über einen CPU-Kern. Die Informationen können für einen gewünschten CPU-Kern mit Hilfe der Funktion F_GetCpuCoreInfo [▶ 93] und mit Hilfe des entsprechenden CPU-Kern-Index ausgelesen werden.

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

Name	Typ	Beschreibung
bRTCore	BOOL	Diese Variable hat den Wert TRUE, falls es sich um einen Echtzeitkern handelt.
bIsolatedCore	BOOL	Diese Variable hat den Wert TRUE, falls es sich um einen isolierten Kern handelt.
nBaseTime	UDINT	Basiszeit des CPU-Kerns, Angabe als Vielfaches von 100ns
nCoreLimit	UDINT	Core-Limit des CPU-Kerns, Angabe in %

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.11	PC oder CX (x86, x64, Arm®)	Tc2_System (System) >= 3.4.24.0

5.13 SYSTEMINFOTYPE

Dieser TwinCAT 2 Datentyp existiert in TwinCAT 3 nicht mehr.
SystemInfoType wurde ersetzt durch PlcAppSystemInfo, einem globalen Systemdatentyp.

5.14 SYSTEMTASKINFOTYPE

Dieser TwinCAT 2 Datentyp existiert in TwinCAT 3 nicht mehr.
SystemTaskInfoType wurde ersetzt durch PlcTaskSystemInfo, einem globalen Systemdatentyp.

5.15 T_AmsNetID

Eine SPS-Variable von diesem Typ ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird. Der String besteht aus sechs, durch Punkte getrennten Zahlenfeldern. Gültige AMS-Netzwerkadressen sind z. B. '1.1.1.2.7.1' oder '200.5.7.170.1.7'. Wird ein Leerstring übergeben, so wird automatisch die AMS-Netzwerkkennung des lokalen Gerätes angenommen.

Namensraum: Tc2_System
Bibliothek: Tc2_System (Tc2_System.compiled-library)

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

5.16 T_AmsNetIdArr

Eine SPS-Variable von diesem Typ enthält die einzelnen Adressbytes der AMS-Netzwerkkennung.

```
TYPE T_AmsNetIdArr : ARRAY[0..5] OF BYTE;
END_TYPE
```

Die Adressbytes besitzen die Netzwerk-Byte-Reihenfolge. D. h. die Adresse '127.16.17.3.1.1' wird in dem Bytearray auf folgende Weise abgebildet:

byte[0] := 127
byte[1] := 16
byte[2] := 17
byte[3] := 3
byte[4] := 1
byte[5] := 1

Beispiel:

Siehe: F_ScanAmsNetIds [▶ 113]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.17 T_AmsPort

Eine Variable von diesem Typ enthält die ADS-Portnummer. ADS-Geräte im TwinCAT-Netzverbund werden durch eine AMS-Netzwerkadresse und eine Portnummer identifiziert. Auf jedem TwinCAT-Einzelsystem sind folgende dezimale Portnummern invariant festgelegt.

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

Tabelle mit festgelegten ADS-Portnummern:

ADS-Gerät	Portnummer
Nockenschaltwerk	900
Laufzeitsystem 1 Laufzeitsystem 2 Laufzeitsystem 3 Laufzeitsystem 4 Laufzeitsystem 5 Laufzeitsystem n	Laufzeitsystem 1: 851 (in TwinCAT 2: 801) Laufzeitsystem 2: 852 (in TwinCAT 2: 811) Laufzeitsystem 3: 853 (in TwinCAT 2: 821) Laufzeitsystem 4: 854 (in TwinCAT 2: 831) Laufzeitsystem 5: 855 Laufzeitsystem n: 850 + n, usw.
NC	500
Reserviert	400
I/O	300
Echtzeitkern	200
Meldesystem (Logger)	100

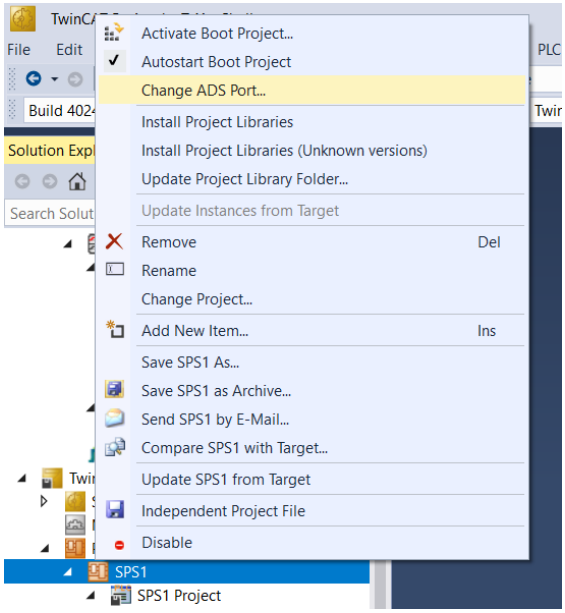
Auf einem TwinCAT2-System können bis zu vier unabhängige SPS-Laufzeitsysteme (SPS-Projekte) laufen, jedes SPS-Projekt ist als eigenständige SPS zu betrachten. Die Portnummer wird, zusammen mit der Netzwerkadresse, beim Aufruf der ADS-Bausteine als Eingangsparameter benötigt.

Für die SPS in TwinCAT3 stehen grundsätzlich die ADS-Ports von 800 bis 899 zur Verfügung. Damit eine Trennung zwischen TwinCAT2- und TwinCAT3-Systemen besteht, empfehlen wir aber, nur die Ports von 851 bis 899 zu nutzen.

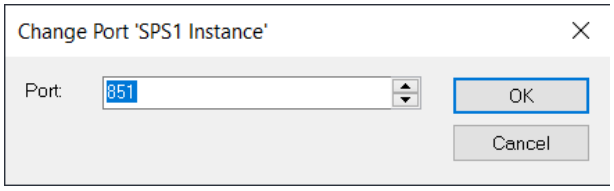
Wenn Sie den ADS-Port mit Hilfe des Dialogs einstellen, wird als kleinster einstellbarer Port der Port 851 angezeigt. Um den Bereich 800-850 zu nutzen, müssen Sie die Portnummer eintippen.

Um die Portnummer über das Dialogfenster einzugeben, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf das gewünschte SPS-Projekt.
2. Klicken Sie **Change ADS Port** an.



⇨ Das Dialogfenster öffnet sich.



3. Wählen Sie die gewünschte Portnummer mit Hilfe der Pfeiltasten aus oder tippen Sie die gewünschte Portnummer ein.
4. Bestätigen Sie die Eingabe mit **OK**.

⇨ Die Portnummer ist im System eingetragen.

Um eine Trennung zwischen TwinCAT2- und TwinCAT3-Systemen zu gewährleisten, empfehlen wir, nur die ADS-Ports von 851 bis 899 zu benutzen.

ADS-Ports außerhalb des Bereiches von 800 bis 899 werden vom Eingabesystem nicht akzeptiert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.18 T_IPv4Addr

Eine Variable von diesem Typ ist ein String mit der (Ipv4) Internet-Protocol-Netzwerkadresse. Z. B. '172.16.7.199'.

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.19 T_IPv4AddrArr

Eine Variable von diesem Typ enthält die einzelnen Adressbytes der (IPv4) Internet-Protocol-Netzwerkadresse.

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

Die Adressbytes haben die Netzwerk-Byte-Reihenfolge. D. h. die Adresse '172.16.7.199' wird in dem Bytearray auf folgende Weise abgebildet:

byte[0] := 172
byte[1] := 16
byte[2] := 7
byte[3] := 199

Beispiel:

Siehe: F_ScanIPv4AddrIds [▶ 92]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.20 T_MaxString

Eine Variable von diesem Typ ist ein String mit maximaler Länge. Längere Strings dürfen benutzt werden, die String-Funktionen können aber zur Zeit nur 255 Zeichen bearbeiten.

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

5.21 TcEvent

TwinCAT EventLogger vs. TwinCAT 3 EventLogger

Der TwinCAT EventLogger wurde durch den Nachfolger TwinCAT 3 EventLogger abgelöst. Der ältere TwinCAT EventLogger wird von TwinCAT 3 bis zur Version 3.1.4024 unterstützt. Neuere TwinCAT-Versionen (>= 3.1.4026.0) unterstützen nur den neueren TwinCAT 3 EventLogger. SPS-Funktionsbausteine hierzu befinden sich in der SPS Bibliothek Tc3_EventLogger.

```
TYPE TcEvent
STRUCT
    Class                : UDINT;
    Prio                 : UDINT;
    Id                   : UDINT;
    bQuitRequired        : BOOL;
    DataFormatStrAddress : PVOID;
    UserFlags            : DWORD;
    Flags                : DWORD;
    StreamType           : UDINT;
    SourceString         : STRING[15]; (* TCEVENT_SRCNAMESIZE *)
    SourceId             : UDINT;
    ProgId               : STRING[31]; (* TCEVENT_FMTPRGSIZE *)
END_STRUCT
END_TYPE
```

Name	Typ	Beschreibung
Class	UDINT	Eventklasse, Wert aus dem Enum E_TcEventClass [▶ 127] entnehmen.
Prio	UDINT	Priorität des Events innerhalb einer Klasse, frei wählbare Zahl (1..MaxUDINT)
Id	UDINT	Id des Events, wird zur eindeutigen Identifizierung im Eventlogger genutzt.
bQuitRequired	BOOL	Flag zum Ein- und Ausschalten der Quittierpflichtigkeit (TRUE → Quittierpflichtig)
DataFormatStrAddress	PVOID	Adresse eines Strings, String enthält Formatieranweisungen (z. B. %d%f formatiert einen Integer- und einen Real-(float)-Wert).
UserFlags	DWORD	32-Bit-Zahl zur freien Verfügung
Flags	DWORD	32-Bit-Zahl zur Kennzeichnung des Events, die Bedeutung der einzelnen Bits sind in den globalen Variablen [▶ 135] der Bibliothek deklariert.
StreamType	UDINT	Typ des Events, Wert aus dem Enum E_TcEventStreamType [▶ 128] entnehmen.
SourceString	STRING	String mit dem Sourcenamen (max. 15 Zeichen [▶ 135])
SourceId	UDINT	Source-ID
ProgId	STRING	String (Prog-Id) mit dem Namen des Formatters (max. 31 Zeichen [▶ 135]). Standard: 'TcEventLogger.TcLogFormatter' oder 'TcEventFormatter.TcXmlFormatter'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0 up to TwinCAT v3.1.4024	PC oder CX (x86, x64, ARM)	Tc2_System (System)

6 Globale Konstanten

6.1 Konstanten

Portnummern

Portnummern	Wert	Beschreibung
AMSPORT_LOGGER	100	Portnummer des Standard-Loggers.
AMSPORT_EVENTLOG	110	Portnummer des TwinCAT Eventloggers.
AMSPORT_R0_RTIME	200	Portnummer des TwinCAT Realtime Servers.
AMSPORT_R0_IO	300	Portnummer des TwinCAT I/O Servers.
AMSPORT_R0_NC	500	Portnummer des TwinCAT NC Servers.
AMSPORT_R0_NCSAF	501	Portnummer des TwinCAT NC Servers (Task SAF).
AMSPORT_R0_NCSVB	511	Portnummer des TwinCAT NC Servers (Task SVB).
AMSPORT_R0_ISG	550	intern
AMSPORT_R0_CNC	600	Portnummer des TwinCAT NC I Servers.
AMSPORT_R0_LINE	700	intern
AMSPORT_R0_PLC	800	Portnummer des TwinCAT PLC Servers (nur auf dem Buscontroller).
AMSPORT_R0_PLC_RTS1	801	Portnummer des TwinCAT 2.xx PLC Servers in der Runtime 1
AMSPORT_R0_PLC_RTS2	811	Portnummer des TwinCAT 2.xx PLC Servers in der Runtime 2
AMSPORT_R0_PLC_RTS3	821	Portnummer des TwinCAT 2.xx PLC Servers in der Runtime 3
AMSPORT_R0_PLC_RTS4	831	Portnummer des TwinCAT 2.xx PLC Servers in der Runtime 4
AMSPORT_R0_CAM	900	Portnummer des TwinCAT CAM Servers.
AMSPORT_R0_CAMTOOL	950	Portnummer des TwinCAT CAMTOOL Servers.
AMSPORT_R3_SYSSERV	10000	Portnummer des TwinCAT System Services..
AMSPORT_R3_SCOPESERVER	14001	Portnummer des TwinCAT Scope Servers

Ads-States

ADS States	Wert	Beschreibung
ADSSTATE_INVALID	0	ADS Status: invalid
ADSSTATE_IDLE	1	ADS Status: idle
ADSSTATE_RESET	2	ADS Status: reset.
ADSSTATE_INIT	3	ADS Status: init
ADSSTATE_START	4	ADS Status: start
ADSSTATE_RUN	5	ADS Status: run
ADSSTATE_STOP	6	ADS Status: stop
ADSSTATE_SAVECFG	7	ADS Status: save configuration
ADSSTATE_LOADCFG	8	ADS Status: load configuration
ADSSTATE_POWERFAILURE	9	ADS Status: Power failure
ADSSTATE_POWERGOOD	10	ADS Status: Power good
ADSSTATE_ERROR	11	ADS Status: Error
ADSSTATE_SHUTDOWN	12	ADS Status: Shutdown
ADSSTATE_SUSPEND	13	ADS Status: Suspend
ADSSTATE_RESUME	14	ADS Status: Resume
ADSSTATE_CONFIG	15	ADS Status: Configuration (System is in config mode)
ADSSTATE_RECONFIG	16	ADS Status: Reconfiguration (System should restart in config mode)
ADSSTATE_STOPPING	17	
ADSSTATE_INCOMPATIBLE	18	
ADSSTATE_EXCEPTION	19	
ADSSTATE_MAXSTATES	20	Max. number of available ads states

ADS/System Dienste

Reserved Index Groups	Wert	Beschreibung
ADSIGRP_SYMTAB	16#F000	
ADSIGRP_SYMNAME	16#F001	
ADSIGRP_SYMVAL	16#F002	
ADSIGRP_SYM_HNDBYNAME	16#F003	
ADSIGRP_SYM_VALBYNAME	16#F004	
ADSIGRP_SYM_VALBYHND	16#F005	
ADSIGRP_SYM_RELEASEHND	16#F006	
ADSIGRP_SYM_INFOBYNAME	16#F007	
ADSIGRP_SYM_VERSION	16#F008	
ADSIGRP_SYM_INFOBYNAMEEX	16#F009	
ADSIGRP_SYM_DOWNLOAD	16#F00A	
ADSIGRP_SYM_UPLOAD	16#F00B	
ADSIGRP_SYM_UPLOADINFO	16#F00C	
ADSIGRP_SYMNOTE	16#F010	
ADSIGRP_IOIMAGE_RWIB	16#F020	
ADSIGRP_IOIMAGE_RWIX	16#F021	
ADSIGRP_IOIMAGE_RISIZE	16#F025	
ADSIGRP_IOIMAGE_RWOB	16#F030	
ADSIGRP_IOIMAGE_RWOX	16#F031	
ADSIGRP_IOIMAGE_RWOSIZE	16#F035	
ADSIGRP_IOIMAGE_CLEARI	16#F040	
ADSIGRP_IOIMAGE_CLEARO	16#F050	
ADSIGRP_IOIMAGE_RWIOB	16#F060	
ADSIGRP_DEVICE_DATA	16#F100	
ADSIOFFS_DEVDATA_ADSSTATE	16#0000	
ADSIOFFS_DEVDATA_DEVSTATE	16#0002	

System Service Dateidienste

System Service Index Groups	Wert	Beschreibung
SYSTEMSERVICE_OPENCREATE	100	
SYSTEMSERVICE_OPENREAD	101	
SYSTEMSERVICE_OPENWRITE	102	
SYSTEMSERVICE_CREATEFILE	110	
SYSTEMSERVICE_CLOSEHANDLE	111	
SYSTEMSERVICE_FOPEN	120	
SYSTEMSERVICE_FCLOSE	121	
SYSTEMSERVICE_FREAD	122	
SYSTEMSERVICE_FWRITE	123	
SYSTEMSERVICE_FSEEK	124	
SYSTEMSERVICE_FTELL	125	
SYSTEMSERVICE_FGETS	126	
SYSTEMSERVICE_FPUTS	127	
SYSTEMSERVICE_FSCANF	128	
SYSTEMSERVICE_FPRINTF	129	
SYSTEMSERVICE_FEOF	130	
SYSTEMSERVICE_FDELETE	131	
SYSTEMSERVICE_FRENAME	132	
SYSTEMSERVICE_REG_HKEYLOCALMACHINE	200	
SYSTEMSERVICE_SENDEMAIL	300	
SYSTEMSERVICE_TIMESERVICES	400	
SYSTEMSERVICE_STARTPROCESS	500	
SYSTEMSERVICE_CHANGENETID	600	

System Service Uhrzeitdienste

System Service Index Offsets (Timeservices)	Wert	Beschreibung
TIMESERVICE_DATEANDTIME	1	
TIMESERVICE_SYSTEMTIMES	2	
TIMESERVICE_RTCTIMEDIFF	3	
TIMESERVICE_ADJUSTTIMETORTC	4	

ADSLOG message types

Masken für die Log-Ausgabe	Wert	Beschreibung
ADSLOG_MSGTYPE_HINT	16#01	
ADSLOG_MSGTYPE_WARN	16#02	
ADSLOG_MSGTYPE_ERROR	16#04	
ADSLOG_MSGTYPE_LOG	16#10	
ADSLOG_MSGTYPE_MSGBOX	16#20	
ADSLOG_MSGTYPE_RESOURCE	16#40	
ADSLOG_MSGTYPE_STRING	16#80	

BOOTDATA flags

Masken für Bootdata-Flags	Wert	Beschreibung
BOOTDATAFLAGS_RETAIN_LOADED	16#01	
BOOTDATAFLAGS_RETAIN_INVALID	16#02	
BOOTDATAFLAGS_RETAIN_REQUESTED	16#04	
BOOTDATAFLAGS_PERSISTENT_LOADED	16#10	
BOOTDATAFLAGS_PERSISTENT_INVALID	16#20	

Masken für BSOD-Flags	Wert	Beschreibung
SYSTEMSTATEFLAGS_BSOD	16#01	BSOD: Blue Screen of Death
SYSTEMSTATEFLAGS_RTVIOLATION	16#02	Echtzeit-Verletzung, Latenzzeitüberschreitung

File open modes

Masken für die File-Ausgabe	Wert	Beschreibung
FOPEN_MODEREAD	16#0001	'r': Öffnet eine Datei zum Lesen
FOPEN_MODEWRITE	16#0002	'w': Öffnet eine Datei zum Schreiben, evtl. bestehende Datei wird überschrieben
FOPEN_MODEAPPEND	16#0004	'a': Öffnet eine Datei zum Schreiben, wird an evtl. bestehende Datei angehängt. Falls Datei noch nicht besteht, wird die Datei angelegt.
FOPEN_MODEPLUS	16#0008	'+': Öffnet eine Datei zum Lesen und Schreiben.
FOPEN_MODEBINARY	16#0010	'b': Öffnet eine Datei zum binären Lesen und Schreiben.
FOPEN_MODETEXT	16#0020	't': Öffnet eine Datei zum textmäßigen Lesen und Schreiben.

Eventlogger constants

Masken für Eventlogger Flags	Wert	Beschreibung
TCEVENTFLAG_PRIOCLASS	16#0010	Klasse und Priorität wird durch Formatter festgelegt
TCEVENTFLAG_FMTSELF	16#0020	Formatierungsinformation kommt mit dem Event
TCEVENTFLAG_LOG	16#0040	Loggen.
TCEVENTFLAG_MSGBOX	16#0080	Messagebox anzeigen.
TCEVENTFLAG_SRCID	16#0100	Verwendung von Source-Id statt Source-Name.

TwinCAT Eventlogger Statusmel-dungen	Wert	Beschreibung
TCEVENTSTATE_INVALID	16#0000	Ungültig, kommt auch wenn Event noch nicht gemeldet.
TCEVENTSTATE_SIGNALED	16#0001	Event ist gemeldet, aber weder gegangen noch quittiert.
TCEVENTSTATE_RESET	16#0002	Event ist abgemeldet ('gegangen').
TCEVENTSTATE_CONFIRMED	16#0010	Event ist quittiert.
TCEVENTSTATE_RESETCON	16#0012	Event ist abgemeldet und quittiert

TwinCAT Eventlogger Status-meldungen	Wert	Beschreibung
TCEVENT_SRCNAMESIZE	15	Max. Länge für den Source-Namen.
TCEVENT_FMTPRGSIZE	31	Max. Länge für den Namen des Formatters.

Andere	Wert	Beschreibung
PI	3.14159265358979323846264338 32795	Pi-Zahl
DEFAULT_ADS_TIMEOUT	T#5s	Default ADS-Timeout
MAX_STRING_LENGTH	255	Max Stringlänge des T_MaxString Datentyps

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

6.2 Bibliotheksversion

Alle Bibliotheken haben eine bestimmte Version. Diese Version ist u. a. im SPS-Bibliotheks-Repository zu sehen. Eine globale Konstante enthält die Information über die Bibliotheksversion:

Global_Version

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

Name	Typ	Beschreibung
stLibVersion_Tc2_System	ST_LibVersion	Versionsnummer der Tc2_System-Bibliothek (Typ: ST_LibVersion)

Um zu sehen, ob die Version, die Sie haben auch die Version ist, die Sie brauchen, benutzen Sie die Funktion F_CmpLibVersion [▶ 91].

Alle anderen Möglichkeiten Bibliotheksversionen zu vergleichen, die Sie von TwinCAT 2 kennen, sind veraltet.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, Arm®)	Tc2_System (System)

7 Beispiele

7.1 Beispiel mit AdsReadInd-/AdsReadRes-Funktionsbausteinen

Das Beispiel zeigt die Implementierung einer einfachen ADS-Server-Applikation in der SPS. Die Server-Applikation kann die ADSREAD-Requests einer ADS-Client-Applikation bearbeiten.

In der Beispielapplikation werden ADSREAD-Requests verwendet um in der SPS-Task eine SPS-Zählervariable zu inkrementieren/dekrementieren oder zurückzusetzen. Beim Erfolg wird der Wert der Zählervariablen an die ADS-Client-Applikation zurück gesendet

Die kompletten Sourcen der ADS-Server-Applikation können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/710574987.zip

Eine zu dem ADS-Server passende ADS-Client-Applikation kann hier gefunden werden: Beispiel mit ADSREAD-Funktionsbaustein [▶ 144].

ADS-Client-Applikation

Der gewünschte Dienst in der SPS-Task wird in dem Index-Group-Parameter verschlüsselt:

IG:0x80000001 → Die Zählervariable inkrementieren;

IG:0x80000002 → Die Zählervariable dekrementieren;

IG:0x80000003 → Die Zählervariable = 0 setzen;

Der Index-Offset-Parameter ist Null.

Damit die Requests an die SPS-Task weiter geleitet werden, muss in dem Index-Group-Parameter das höchstwertige Bit gesetzt werden.

SPS-Programm

Die ADSREAD-Requests werden in der SPS-Task von einer Instanz des ADSREADIND [▶ 28]-Funktionsbausteins als Indications abgefangen. Danach werden die Parameter Index-Group, Index-Offset und die angeforderte Datenlänge auf Gültigkeit überprüft. In der CASE-Anweisung wird die gewünschte Operation an der SPS-Variablen durchgeführt. Beim Erfolg wird ein Response von einer Instanz des ADSREADRES [▶ 37]-Funktionsbausteins an den Aufrufer mit dem aktuellen Wert der SPS-Variablen zurückgesendet. Im Fehlerfall eine entsprechende Fehlermeldung. Im nächsten Zyklus werden die Flags CLEAR und RESPOND an den Funktionsbausteinen zurückgesetzt um weitere Indications verarbeiten zu können.

Deklarationsteil

```
PROGRAM MAIN
VAR
    fbReadInd     : ADSREADIND;(* Indication function block instance *)
    fbReadRes     : ADSREADRES;(* Response function block instance *)
    sNetId        : T_AmsNetID;
    nPort         : T_AmsPort;
    nInvokeId     : UDINT;
    nIdxGrp       : UDINT;
    nIdxOffs      : UDINT;
    cbLength      : UDINT;(* Requested read data/buffer byte size *)
    cbRead        : UDINT;(* Returned read data/buffer byte size *)
    pRead         : PVOID;(* Pointer to returned read data/buffer *)
    nErrID        : UDINT;(* Read indication result error code *)
    nCounter      : INT;(* Server data *)
END_VAR
```

Implementierung

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

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

      cbRead := 0;
      pRead := 0;
      nErrID     := DEVICE_SRVNOTSUPP;


     CASE nIdxGrp OF
          (*-----------------------------------------------------------------------------------*)
          16#80000001:
               CASE nIdxOffs OF
                    0:(* Increment counter value *)
                         IF cbLength >= SIZEOF(nCounter) THEN
                              nCounter := nCounter + 1;
                              cbRead := SIZEOF(nCounter);
                              pRead := ADR(nCounter);
                              nErrID := NOERR;
                         ELSE (* ADS error (example): Invalid size *)
                              nErrID := DEVICE_INVALIDSIZE;
                         END_IF
               ELSE (* ADS error (example): Invalid index offset *)
                    nErrID := DEVICE_INVALIDOFFSET;
               END_CASE
          (*-----------------------------------------------------------------------------------*)
          16#80000002:
               CASE nIdxOffs OF
                    0:(* Decrement counter value *)
                         IF cbLength >= SIZEOF(nCounter) THEN
                              nCounter := nCounter - 1;
                              cbRead := SIZEOF(nCounter);
                              pRead := ADR(nCounter);
                              nErrID := NOERR;
                         ELSE(* ADS error (example): Invalid size *)
                              nErrID := DEVICE_INVALIDSIZE;
                         END_IF
               ELSE (* ADS error (example): Invalid index offset *)
                    nErrID := DEVICE_INVALIDOFFSET;
               END_CASE
          (*-----------------------------------------------------------------------------------*)
          16#80000003:
               CASE nIdxOffs OF
                    0:(* Reset counter value *)
                         IF cbLength >= SIZEOF(nCounter) THEN
                              nCounter := 0;
                              cbRead := SIZEOF(nCounter);
                              pRead := ADR(nCounter);
                              nErrID := NOERR;
                         ELSE(* ADS error (example): Invalid size *)
                              nErrID := DEVICE_INVALIDSIZE;
                         END_IF
               ELSE (* ADS error (example): ervice is not supported by server *)
                    nErrID := DEVICE_SRVNOTSUPP;
               END_CASE

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

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

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

7.2 Beispiel mit AdsWriteInd-/AdsWriteRes-Funktionsbausteinen

Das Beispiel zeigt die Implementierung einer einfachen ADS-Server-Applikation in der SPS. Die Server Applikation kann auf die ADSWRITE-Requests einer ADS-Client-Applikation bearbeiten.

In der Beispielapplikation werden ADSWRITE-Requests verwendet um Array mit Integer-Werten in die SPS-Task zu übertragen. Die empfangenen Daten werden in der SPS in eine entsprechende Array-Variable kopiert.

Die kompletten Sourcen der ADS-Server-Applikation können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/710582923.zip

Eine zu dem ADS-Server passende ADS-Client-Applikation kann hier gefunden werden: Beispiel mit ADSWRITE-Funktionsbaustein [▶ 145].

ADS-Client-Applikation

Der gewünschte Dienst/Befehl in der SPS-Task wird in dem Index-Group und Index-Offset Parameter verschlüsselt. Z.B.:

IG:0x80000005 und IO:0x00000007→ Die gesendeten Daten in den Array in der SPS kopieren.

> Damit die Requests an die SPS-Task weiter geleitet werden, muss in dem Index-Group-Parameter das höchstwertige Bit gesetzt werden.

SPS-Programm

Die Requests werden in der SPS-Task von einer Instanz des ADSWRITEIND [▶ 31]-Funktionsbausteins als Indications abgefangen. Danach werden die Parameter Index-Group, Index-Offset und die gesendete Datenlänge auf Gültigkeit überprüft und die gewünschte Operation an der SPS-Variablen durchgeführt. Als Nächstes wird ein Response von einer Instanz des ADSWRITERES [▶ 38]-Funktionsbausteins an den Aufrufer (eventuell mit einem Fehlercode) zurückgesendet. Im nächsten Zyklus werden die Flags CLEAR und RESPOND rückgesetzt um weitere Indications verarbeiten zu können.

> Mit der steigenden Flanke am CLEAR-Eingang des ADSWRITEIND-Funktionsbausteins wird der Adresspointer auf die zuletzt gesendeten Daten ungültig (== NULL).
>
> Aus diesem Grund werden die gesendeten Daten zuerst in die SPS-Variable kopiert und dann der CLEAR-Eingang auf TRUE gesetzt.

Deklarationsteil

```
PROGRAM MAIN
VAR
   fbWriteInd  : ADSWRITEIND;
   fbWriteRes  : ADSWRITERES;
   sNetId      : T_AmsNetID;
   nPort       : T_AmsPort;
   nInvokeId   : UDINT;
   nIdxGrp     : UDINT;
   nIdxOffs    : UDINT;
   cbWrite     : UDINT;(* Byte size of written data *)
   pWrite      : PVOID;(* Pointer to written data buffer *)
   nResult     : UDINT;(* Write indication result error code *)
   arrInt      : ARRAY[0..9] OF INT;(* Server data *)
END_VAR
```

Implementierung

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

   sNetId       := fbWriteInd.NETID;
```

```
   nPort        := fbWriteInd.PORT;
   nInvokeId    := fbWriteInd.INVOKEID;
   nIdxGrp      := fbWriteInd.IDXGRP;
   nIdxOffs     := fbWriteInd.IDXOFFS;
   cbWrite      := fbWriteInd.LENGTH;
   pWrite       := fbWriteInd.DATAADDR;
   nResult      := DEVICE_SRVNOTSUPP;

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

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

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

7.3 Beispiel mit AdsRead-Funktionsbaustein

Das Beispiel demonstriert die Verwendung des ADSREAD-Funktionsbausteins in einer ADS-Client-Applikation.

Die kompletten Sourcen der ADS-Client-Applikation können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/710578827.zip

Deklarationsteil

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

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

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

Implementierung

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

            fbReadReq( READ := FALSE );

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

```
            ELSIF bDecrement THEN(* Decrement counter value *)
                bDecrement := FALSE;
                fbReadReq( IDXGRP := 16#80000002, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            ELSIF bReset THEN(* Reset counter value *)
                bReset := FALSE;
                fbReadReq( IDXGRP := 16#80000003, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            ELSIF bOther THEN(* Call unsupported function *)
                bOther := FALSE;
                fbReadReq( IDXGRP := 16#80000004, IDXOFFS := 0, LEN := SIZEOF(nCounter), DESTADDR :=
 ADR(nCounter), READ := TRUE );
            END_IF

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

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

END_CASE
```

7.4 Beispiel mit AdsWrite-Funktionsbaustein

Das Beispiel demonstriert die Verwendung des ADSWRITE-Funktionsbausteins in einer ADS-Client-Applikation.

Die kompletten Sourcen der ADS-Client-Applikation können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/710586763.zip

Deklarationsteil

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

Implementierung

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


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

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

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

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

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

END_CASE
```

7.5 EventLogger-Meldungen aus der SPS senden/quittieren

Das Beispiel zeigt die Verwendung des ADSLOGEVENT-Funktionsbausteins.

Die kompletten Sourcen der Beispielapplikation können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/711421451.zip

Schrittweiser Ablauf

Konfigurieren eines Events:

Struktur EventConfigData [▶ 133] parametrieren

Übergeben von Parametern:

Adresse auf eine Struktur, ein Array oder eine Einzelvariable mit ADR Operator an EventDataAddress anlegen. Länge der Struktur, des Array oder der Einzelvariablen mit SIZEOF Operator ermitteln und an Eingang EventDataLength anlegen. Soll z. B. eine Struktur mit einer INT und einer LREAL Variablen mit dem Event übergeben werden, so ist eine Struktur mit diesen beiden Komponenten zu erstellen und zu instanziieren. Die Adresse und die Länge dieser Instanz muss übergeben werden.

Setzen eines Events:

Steigende Flanke am Event-Eingang

Rücksetzen eines Events:

Fallende Flanke am Event-Eingang

Quittieren eines Events:

Steigende Flanke am Quit-Eingang

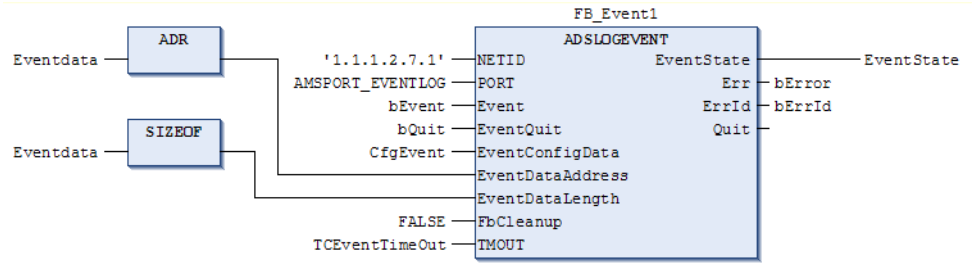
Komplettes Löschen der Instanz:

Mit der steigenden Flanke am Eingang FbCleanup wird der Inhalt der Instanz komplett gelöscht. Damit wird nicht unmittelbar ein bestehendes Event aus dem EventLogger gelöscht.

Nachdem ein Event an den EventLogger verschickt wurde, ändert sich der Status des Event [▶ 135] sichtbar am Eventstate-Ausgang.

Aufruf des FBs ADSLOGEVENT

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



Deklarationsteil

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

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

Implementierung

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

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

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

7.6 Dateizugriff aus der SPS

In diesem Beispiel wird die Anwendung der SPS-Funktionsbausteine für den Dateizugriff aus der Tc2_System-Bibliothek vorgestellt. Mit Hilfe der vorhandenen Funktionsbausteine wurde ein neuer Funktionsbaustein FB_FileCopy realisiert. Mit dem FB_FileCopy-Funktionsbaustein können Binärdateien auf dem lokalen TwinCAT System oder zwischen einem lokalen und einem remote-TwinCAT-System kopiert werden.

Die kompletten Sourcen zu dem Beispielprojekt können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/707895179.zip

Mit den FB_FileCopy-Funktionsbaustein kann nicht auf Netzwerklaufwerke zugegriffen werden.

Bei einer steigenden Flanke am bExecute-Eingang des FB_FileCopy-Bausteines werden folgende Schritte ausgeführt:

a) Öffnen der Quell- und Ziel-Datei

b) Lesen der Quell-Datei in einen Puffer

c) Schreiben der gelesenen Bytes aus dem Puffer in die Ziel-Datei

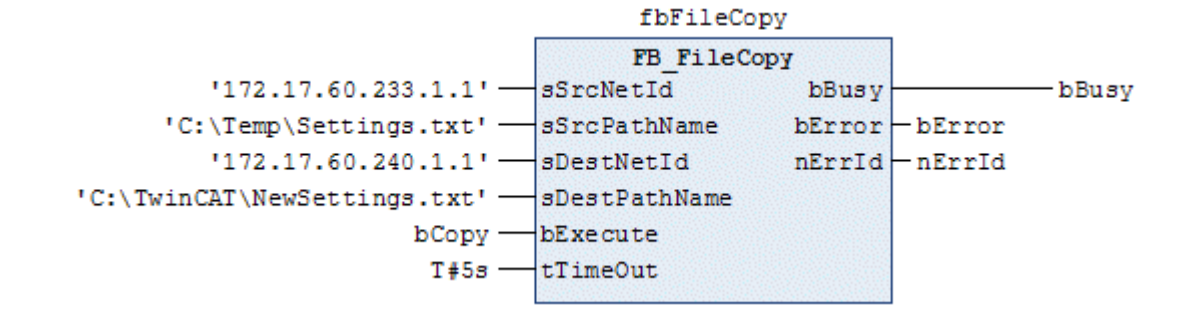
d) Überprüfen ob das Ende der Quell-Datei erreicht wurde. Wenn nicht dann b) und c) wiederholen. Wenn ja, dann zu e) springen

e) Schließen der Quell- und Ziel-Datei

Die Datei wird stückweise kopiert. Die Größe des Puffers wurde im Beispiel auf 1000 Byte festgelegt, kann aber geändert werden.

SPS-Programm

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



Deklarationsteil

```
FUNCTION_BLOCK FB_FileCopy
VAR_INPUT
    sSrcNetId            : T_AmsNetId;
    sSrcPathName         : T_MaxString;
    sDestNetId           : T_AmsNetId;
    sDestPathName        : T_MaxString;
    bExecute             : BOOL;
    tTimeOut             : TIME := DEFAULT_ADS_TIMEOUT;
END_VAR
VAR_OUTPUT
    bBusy                : BOOL;
    bError               : BOOL;
    nErrId               : UDINT;
END_VAR
VAR
    fbFileOpen           : FB_FileOpen;
    fbFileClose          : FB_FileClose;
    fbFileRead           : FB_FileRead;
    fbFileWrite          : FB_FileWrite;
    hSrcFile             : UINT := 0;(* File handle of the source file *)
    hDestFile            : UINT := 0;(* File handle of the destination file *)

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

Implementierung

```
RisingEdge(CLK:=bExecute);

CASE Step OF
     0:     (* Idle state *)
          IF RisingEdge.Q THEN
               bBusy := TRUE;
               bError:= FALSE;
               nErrId:=0;
```

```
            Step := 1;
            cbReadLength:=0;
            hSrcFile:=0;
            hDestFile:=0;
        END_IF

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

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

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

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

```
                    ELSE
                         Step := 5; (* Repeat reading/writing *)
                    END_IF
               END_IF
          END_IF

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

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

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

END_CASE
```

7.7 Testen der CPU Reserve eines CX70xx

Dieses Beispiel ist ein Testprojekt zum Testen der CPU Reserve eines CX70xx. Der enthaltene Baustein `FB_Test_CPU_Performance` misst die CPU-Reserve, die Sie mit ihrer Applikation haben. Der Baustein liest die aktuelle CPU-Leistung und Zykluszeit aus. Der Baustein steigert dann die CPU-Last so lange, bis der CX7k nicht mehr in Echtzeit arbeitet. Dann reduziert er die Last so lange wieder, bis eine stabile Echtzeit erreicht wird. Der Baustein ermittelt dann die CPU-Leistung und die Zykluszeit und verrechnet sie mit der Zeit und Last, die am Anfang der Messung gemacht wurde und gibt Ihnen das Delta an. Verwenden Sie den Baustein nur zu Testzwecken und nicht in einer realen Umgebung.

Wenn die CPU-Reserve größer als 20% ist, können Sie die Task-Zykluszeit schneller machen als die aktuell verwendete. Vorteile einer schnelleren Task sind das schnelleres Reagieren auf Eingänge und je nach Programminhalt eine schnellere Applikation. Wenige Millisekunden können in der Summe den Output einer Maschine erhöhen. Eine Reserveleistung von 20 % ist ideal.

Die Sourcen zu dem Beispielprojekt können hier entpackt werden: https://infosys.beckhoff.com/content/1031/TcPlcLib_Tc2_System/Resources/11298888331.zip

Verfälschtes Messergebnis

Wenn der Baustein in einer niederprioren Task verwendet wird, ist das Ergebnis verfälscht und die realen Daten können nicht ermittelt werden. Durch das lange Messen werden auch die langsamen Tasks mit berücksichtigt.

- Verwenden Sie den Baustein möglichst immer in der schnellsten hochprioren Task.

Nicht durchlaufene Schleifen oder Programmteile während der Messung verfälschen die Messung

Schleifen, mehrere Tasks und damit stark schwankende Zykluszeiten verursachen eine stark schwankende CPU-Auslastung.

- Stellen Sie den TimeOut des Bausteins auf einen größeren Wert, weil der Baustein sich die höchste CPU-Auslastung sucht und dann länger braucht als bei einer konstanten CPU-Auslastung.

Abbruch der Messung

Der Baustein kann nur verwendet werden, wenn es keine Echtzeitverletzungen in Ihrer Konfiguration gibt. Sollte der Baustein schon beim Starten Echtzeitverletzungen auslesen, bricht der Baustein die Messung ab.

Informationen zum Baustein `FB_Test_CPU_Performance`:

VAR_INPUT

Name	Type	Beschreibung
bExecute	BOOL	Positive Flanke aktiviert den Baustein.
tTimeOut	TIME	Zeit, die bei einer Überschreitung die Messung abbricht.

VAR_OUTPUT

Name	Type	Beschreibung
bBusy	BOOL	Der Baustein ist aktiv und arbeitet.
bError	BOOL	Der Baustein hat einen Fehler.
nErrorID	UDINT	ADS Fehler Code
nCpuLoadReserve	UDINT	Reserve der CPU in [%]
fCycleTimeReserve	LREAL	Reserve der Zykluszeit in [ms]

Entwicklungsumgebung	Zielplattform	Einzubindende SPS- Bibliotheken
TwinCAT v3.1.4024.22	PC oder CX (x86, x64, ARM)	Tc2_System (System) >= 3.4.25.0

8 Anhang

8.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 152]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 152]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 153]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 155]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet, nicht erreichbar oder nicht installiert.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet. Mehrere Ursachen sind möglich. Beispielsweise beim Anlegen von Routen, dass ein falsches Passwort angegeben wurde.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert. Mehrere Ursachen sind möglich. Beispielsweise, dass eine Unidirectionale ADS Route in die umgekehrte Richtung verwendet wird.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.

Hex	Dec	HRESULT	Name	Beschreibung
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

8.2 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Trademark statements

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Third-party trademark statements

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Mehr Informationen:
www.beckhoff.com/te1000

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