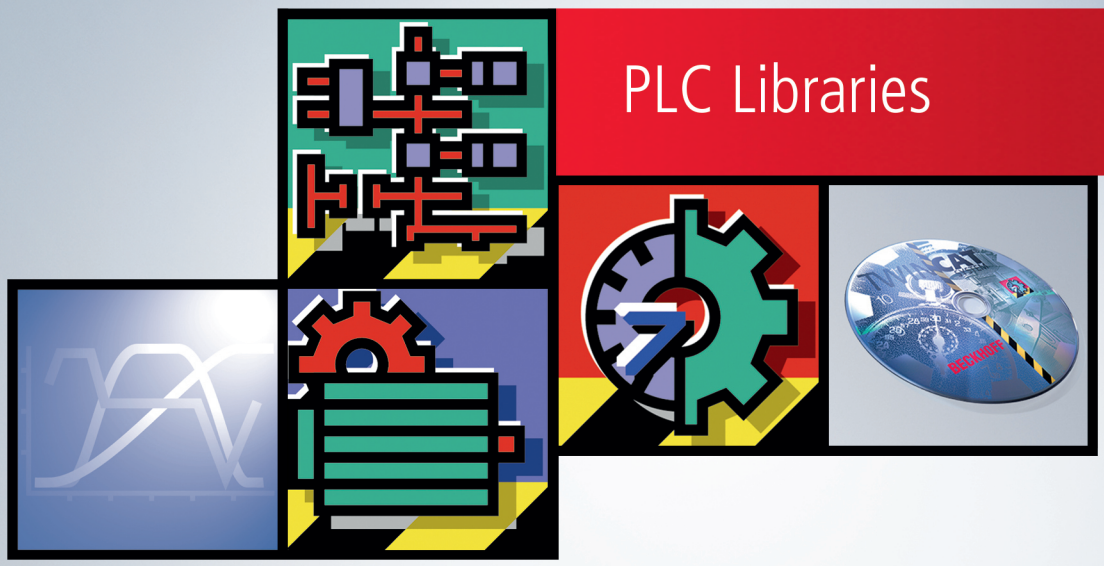
BECKHOFF New Automation Technology

Handbuch | DE

TX1200

TwinCAT 2 | PLC-Bibliothek: TcSystem



11.10.2022 | Version: 1.1

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche 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 weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

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

Patente

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

EtherCAT®

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

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

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.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

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 System-Bibliothek enthält solche Funktionen und Funktionsbausteine des TwinCAT Systems, die nicht zum Standardumfang der IEC61131-3 gehören und dementsprechend herstellerspezifisch sind.

Funktionsbausteine

ADS-Funktionsbausteine

Name	Beschreibung
ADSREAD [▶ 14]	Lesen von Daten über ADS.
ADSWRITE [▶ 15]	Schreiben von Daten über ADS.
ADSRDWRT [▶ 17]	Schreiben und Lesen von Daten über ADS
ADSRDSTATE [▶ 19]	Lesen des Status eines Gerätes über ADS.
ADSWRTCTL [▶ 20]	Schreiben von Control-Wörtern an ein Gerät über ADS.
ADSRDDEVINFO [▶ 22]	Lesen der Geräteinformation über ADS.
ADSREADEX [▶ 23]	Lesen von Daten über ADS und Ermittlung der tatsächlichen Anzahl der gelesenen Bytes.
ADSRDWRTEX [▶ 25]	Schreiben und Lesen von Daten über ADS und Ermittlung der tatsächlichen Anzahl der gelesenen Bytes.

Erweiterte ADS-Funktionsbausteine

Name	Beschreibung
ADSREADIND [▶ 27]	ADSREAD-Indication
ADSWRITEIND [▶ 28]	ADSWRITE-Indication
ADSRDWRTIND [▶ 30]	ADSRDWRT-Indication
ADSREADRES [▶ 31]	ADSREAD-Response
ADSWRITERES [▶ 32]	ADSWRITE-Response
ADSRDWRTRES [▶ 33]	ADSRDWRT-Response

Generelle Funktionsbausteine

Name	Beschreibung
DRAND [▶ 38]	Zufallszahlengenerator
GETCURTASKINDEX [▶ 39]	Ermitteln des aktuellen Taskindex.

Zeitfunktionsbausteine

Name	Beschreibung
GETSYSTEMTIME [▶ 40]	Den Betriebssystem-Zeitstempel auslesen.
GETTASKTTIME [▶ 41]	Die (Soll-)Startzeit der Task auslesen.
GETCPUCOUNTER [▶ 42]	Den Zyklusticker der CPU auslesen.
GETCPUACCOUNT [▶ 42]	Den Zyklusticker der SPS-Task auslesen.

Watchdog Funktionsbausteine

Name	Beschreibung
FB_PcWatchdog [▶ 43]	Aktiviert oder Deaktiviert den PC Watchdog. **Ist nur verfügbar auf IPCs mit den Mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051 !**

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_FileOpen [▶ 50]	Öffnen einer Datei.
FB_FileClose [▶ 46]	Schließen einer Datei.
FB_FileWrite [▶ 60]	Schreiben in eine Datei.
FB_FileRead [▶ 54]	Lesen aus einer Datei.
FB_FileSeek [▶ 57]	Verstellen des Dateizeigers.
FB_FileTell [▶ 59]	Die aktuelle Position des Dateizeigers ermitteln.
FB_FilePuts [▶ 53]	Einen Nullterminierten-String in eine Text-Datei schreiben.
FB_FileGets [▶ 49]	Einen String aus einer Text-Datei lesen.
FB_EOF [▶ 45]	Auf Dateiende testen.
FB_FileDelete [▶ 47]	Löschen einer Datei.
FB_FileRename [▶ 56]	Umbenennen einer Datei.
FB_CreateDir [▶ 65]	Erstellt ein neues Verzeichnis.
FB_RemoveDir [▶ 67]	Löscht ein Verzeichnis.

TwinCAT 2.7 Funktionsbausteine für Dateizugriffe

Diese Funktionsbausteine sind nur aus Kompatibilitätsgründen in der Bibliothek enthalten und sollten in neuen Projekten nicht benutzt werden.

Name	Beschreibung
FILEOPEN [▶ 68]	Öffnen einer Datei.
FILECLOSE [▶ 69]	Schließen einer Datei.
FILEWRITE [▶ 71]	Schreiben in eine Datei.
FILEREAD [▶ 72]	Lesen aus einer Datei.
FILESEEK [▶ 73]	Verstellen des Dateizeigers.

IEC SFC Funktionen

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

Name	Beschreibung
SFCActionControl [▶ 75]	Ermöglicht die Benutzung der IEC-Schritte.
AnalyzeExpression [▶ 75]	Wird bei der Benutzung der SFC-Flags benötigt.
AnalyzeExpressionCombined [▶ 76]	Wird bei der Benutzung der SFC-Flags benötigt.
AppendErrorString [▶ 76]	Wird bei der Benutzung der SFC-Flags benötigt um Strings mit Fehlerbeschreibung zu formatieren.

TcEvent Logger 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 [▶ 77]	Absenden und Quittieren von Meldungen zum TwinCAT Eventlogger.
ADSCLEAREVENTS [▶ 80]	Absenden und Quittieren von Meldungen zum TwinCAT Eventlogger.

Weitere Informationen finden Sie in der TwinCAT Event Logger Dokumentation.

Funktionen

Generelle Funktionen

Name	Beschreibung
F_SplitPathName [▶ 82]	Zerlegt den Pfadnamen in vier Einzelkomponenten.
F_CreateIPv4Addr [▶ 83]	Konvertiert einzelne IPv4-Adressbytes in einen String.
F_ScanIPv4Addrlds [▶ 84]	Konvertiert IPv4-Adressstring in einzelne Adressbytes.
SETBIT32 [▶ 84]	Setzen eines Bits in einem DWORD.
CSETBIT32 [▶ 85]	Setzen/rücksetzen eines Bits in einem DWORD.
GETBIT32 [▶ 86]	Lesen eines Bits aus einem DWORD.
CLEARBIT32 [▶ 87]	Löschen eines Bits in einem DWORD.
LPTSIGNAL [▶ 87]	Ausgabe eines Signals auf einem Pin des parallelen Port.
F_GetStructMemberAlignment [▶ 88]	Liest Informationen über die verwendete Speicherausrichtung (alignment)
F_GetVersionTcSystem [▶ 91]	Versionsinformationen abrufen.

I/O Portzugriff

Name	Beschreibung
F_IOPortRead [▶ 91]	Liest I/O Port
F_IOPortWrite [▶ 92]	Beschreibt I/O Port

ADS-Funktionen

Funktionen die mit Hilfe 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 [▶ 94]	Ausgabe einer DINT Variablen ins Logbuch und/oder Messagebox.
ADSLOGLREAL [▶ 96]	Ausgabe einer (L)REAL Variablen ins Logbuch und/oder Messagebox.
ADSLOGSTR [▶ 97]	Ausgabe einer STRING Variablen ins Logbuch und/oder Messagebox.
F_CreateAmsNetId [▶ 98]	Generiert einen formatierten AmsNetId-String.
F_ScanAmsNetIds [▶ 99]	Konvertiert einen AmsNetId-String in einzelnen Adressbytes.

MEMORY-Funktionen

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

Bemerkung

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

Name	Beschreibung
MEMCMP [▶ 100]	Die Werte der Variablen in zwei Speicherbereichen vergleichen
MEMCPY [▶ 101]	Die Variablenwerte von einem Speicherbereich in einen anderen kopieren
MEMSET [▶ 102]	Die Variablen in einem Speicherbereich auf einen bestimmten Wert setzen
MEMMOVE [▶ 103]	Die Variablen von überlappenden Speicherbereichen kopieren

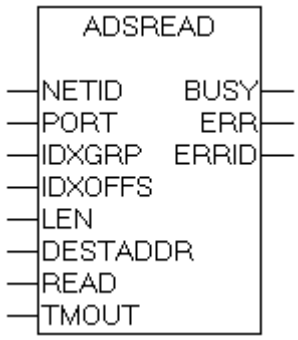
Charakter-Funktionen

Name	Beschreibung
F_ToCHR [▶ 104]	Konvertiert ASCII-Code in ein Charakterzeichen
F_ToASC [▶ 105]	Konvertiert Charakterzeichen in den ASCII-Code

3 Funktionsbausteine

3.1 Ads Funktionsbausteine

3.1.1 ADSREAD



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

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

LEN : Enthält die Anzahl der zu lesenden Daten in Bytes.

DESTADDR : Enthält die 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 : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

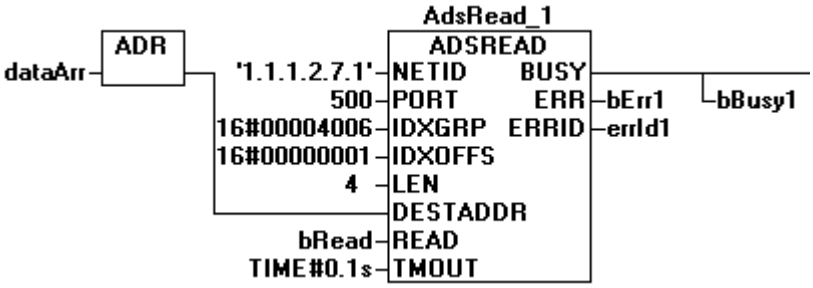
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen 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:

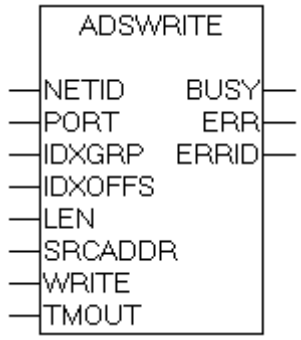


Hierbei wird der Fehlerstatus der Achse Nr. 6 als 4 Byte großes Element abgefragt und in den Puffer dataArr geschrieben. Der IDXGRP 00004006(hex) und der IDXOFFS 00000001(hex) ergeben sich aus der NC-ADS-Dokumentation.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.2 ADSWRITE



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

VAR_INPUT

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

```
    LEN         : UDINT;
    SRCADDR     : DWORD;
    WRITE       : BOOL;
    TMOUT       : TIME;
END_VAR
```

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

LEN : Enthält die Anzahl der zu lesenden Daten in Bytes.

SRCADDR : Enthält die 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 : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

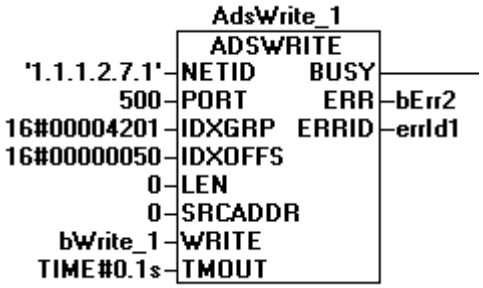
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen 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:

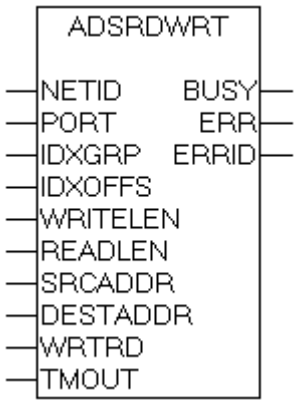


Hierbei wird die NC-Achse Nr. 1 durch einen Schreibbefehl mit IDXGRP 00004201(hex) und der IDXOFFS 00000050(hex) deaktiviert. Zur Aktivierung der Achse müsste ein erneuter Schreibbefehl mit dem IDXOFFS 00000051(hex)) erfolgen. Da dieser Schreibbefehl keine Parameter benötigt, sind die Eingänge LEN und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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.3 ADSRDWRT



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

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

WRITELEN : Enthält die Anzahl der zu schreibenden Daten in Bytes.

READLEN : Enthält die Anzahl der zu lesenden Daten in Bytes.

SRCADDR : Enthält die 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 : Enthält die 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 : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

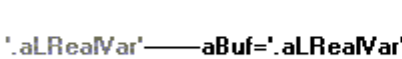
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

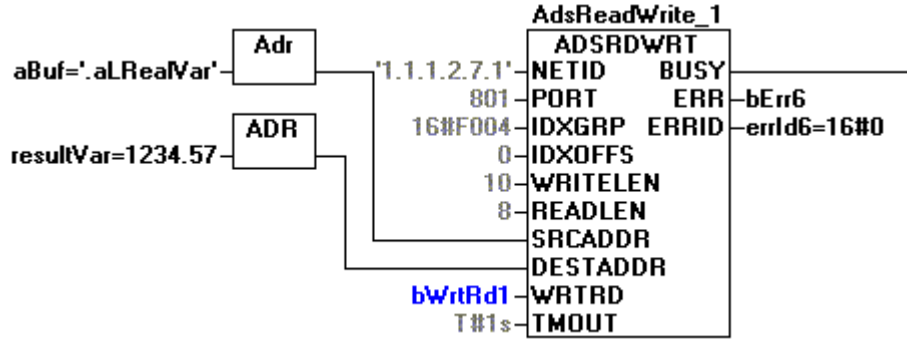
BUSY : Dieser Ausgang bleibt solange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der, an dem 'Timeout'-Eingang angelegten, Zeit. Während Busy = TRUE wird an den Eingängen kein neuer Befehl angenommen. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen 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:



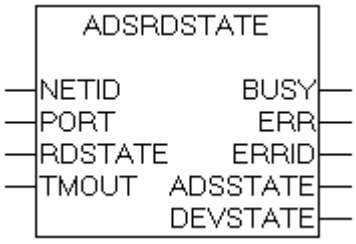


Hierbei soll der Wert der Variablen mit dem Namen ‚aLRealVar' aus der SPS, die auf dem Rechner mit der Net-Id ‚1.1.1.2.7.1' läuft, gelesen werden. Dazu wird die erwähnte Rechneradresse sowie die Portnummer des ersten Laufzeitsystems der SPS, der Index-Group, Index-Offset für namentliches Variablen lesen (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
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.4 ADSRDSTATE



Der Baustein fordert den Zustand eines ADS-Gerätes an.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

RDSTATE : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

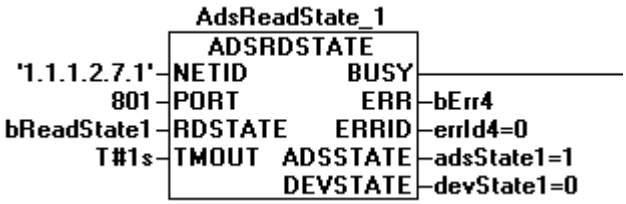
ADSSTATE :

Enthält die 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;

DEVSTATE : Enthält die spezifische Zustandskennzahl des ADS-Zielgerätes. Die hier zurück gelieferten Codes sind Zusatzinformationen, die für das ADS-Gerät spezifisch sind.

Beispiel für den Aufruf des Bausteins in FBD:

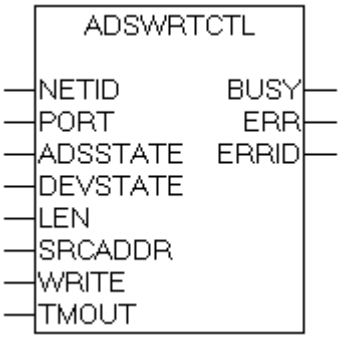


In dem Beispiel wird das SPS-Laufzeitsystem 1 (PortNr.801) auf dem Rechner mit der Netzwerkadresse ‚1.1.1.2.7.1' nach seinem Zustand gefragt. Die Antwort ist adsState = 1 (IDLE) ohne Zusatzcode devState=0.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.5 ADSWRTCTL



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

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

ADSSTATE : Enthält die 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;

DEVSTATE : Enthält die spezifische Zustandskennzahl des ADS-Zielgerätes. Die hier angegebenen Codes sind Zusatzinformationen, die für das ADS-Gerät spezifisch sind.

LEN : Enthält die Anzahl der zu schreibenden Daten in Bytes.

SRCADDR : Enthält die 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 : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

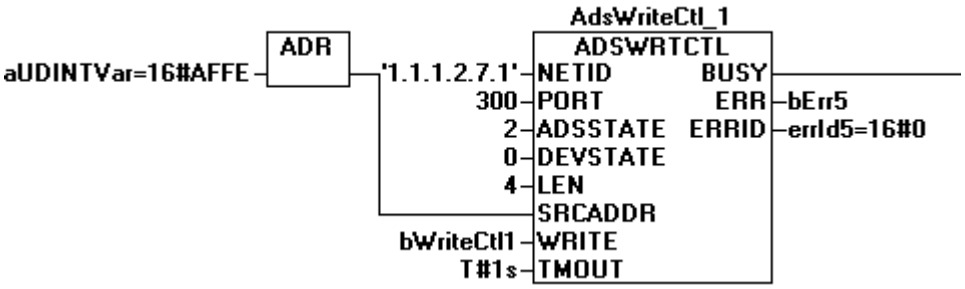
```
VAR_OUTPUT
    BUSY        : BOOL;
    ERR         : BOOL;
    ERRID       : UDINT;
END_VAR
```

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen 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:

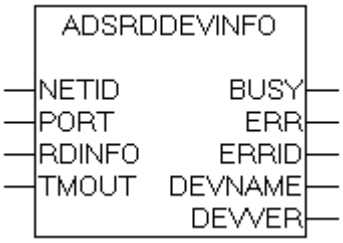


In dem Beispiel wird ein Reset-Kommando (ADSSTATE=2) an den IO-Server (Port300) gesendet, mit den Zusatzdaten hex.AFFE. Der IO-Server führt daraufhin ein Bus-Reset aus.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.6 ADSRDDEVINFO



Der Baustein liest die allgemeinen Geräteinformationen.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

RDINFO : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

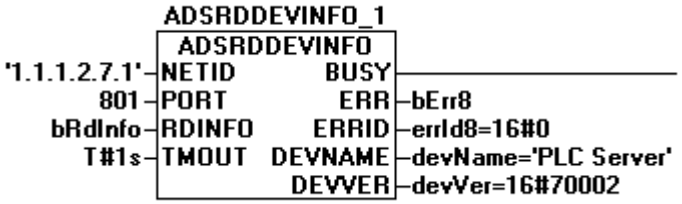
ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

DEVNAME : Enthält den Namen des ADS-Gerätes.

DEVVER : Enthält die Versionsnummer des ADS-Gerätes.

Beispiel für den Aufruf des Bausteins in FBD:

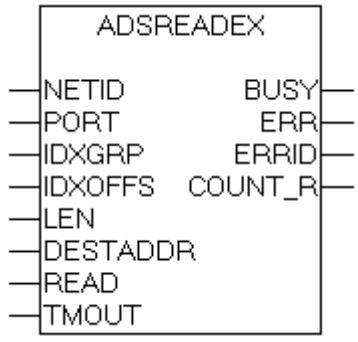


In dem Beispiel werden die Geräteinformationen des ersten SPS-Laufzeitsystems (Port 801) auf dem Rechner ‚1.1.1.2.7.1' gelesen. Als Ergebnis erhält man den Namen ‚PLC Server' sowie die Versionsnummer 02.00.7 .

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.7 ADSREADEX



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.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

LEN : Enthält die Anzahl der zu lesenden Daten in Bytes.

DESTADDR : Enthält die 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 : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

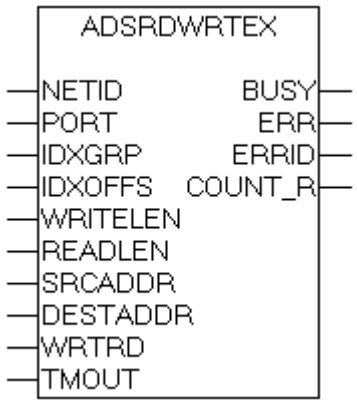
ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

COUNT_R: Anzahl der erfolgreich gelesenen Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.1.8 ADSRDWRTEX



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.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer [▶ 108] des ADS-Gerätes.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes. Dieser Wert ist der ADS-Tabelle des angesprochenen Gerätes zu entnehmen.

WRITELEN : Enthält die Anzahl der zu schreibenden Daten in Bytes.

READLEN : Enthält die Anzahl der zu lesenden Daten in Bytes.

SRCADDR : Enthält die 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 : Enthält die 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 : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

COUNT_R: Anzahl der erfolgreich gelesenen Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2 Erweiterte ADS-Funktionsbausteine

Die erweiterten ADS-Funktionsbausteine ermöglichen den Aufbau einer Client-Server-Kommunikation zwischen einem ADS-Gerät und einer SPS-Task. Bei dem ADS-Gerät kann 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 darauf 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.
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.

Tab. 1: Erweiterte ADS-Funktionsbausteine

Dienst	Name	Beschreibung
READ	ADSREADIND [▶ 27]	ADSREAD-Indication.
	ADSREADRES [▶ 31]	ADSREAD-Response
WRITE	ADSWRITEIND [▶ 28]	ADSWRITE-Indication

Dienst	Name	Beschreibung
	ADSWRITERES [▶ 32]	ADSWRITE-Response
READ & WRITE	ADSRDWRTIND [▶ 30]	ADS-READ & WRITE-Indication
	ADSRDWRTRES [▶ 33]	ADS-READ & WRITE-Response

Die ADS-Geräte werden über eine Portadresse und eine Netzwerkadresse (NETID) adressiert. Die Zieladresse der SPS-Task wird auf folgende Weise gebildet:
Portadresse der SPS-Task = Portnummer des Laufzeitsystems + Tasknummer die angesprochen werden soll
Die aktuelle Tasknummer kann mit dem GETCURTASKINDEX-Funktionsbaustein ermittelt werden.

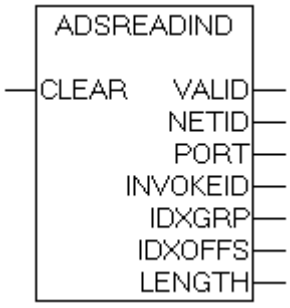
Beispiel:

Die erste SPS-Task des ersten SPS-Laufzeitsystems auf einem lokalen Rechner soll angesprochen werden. Die Portnummer der SPS-Task ergibt sich aus:
PORT = 801 + 1 = 802
NETID = '' (Leerstring)

Bemerkungen:

- Damit ein Request an die SPS-Task weitergeleitet wird, muss in dem IndexGroup-Parameter beim Request das höchstwertige Bit gesetzt werden z.B. IG:=0x80000001.
- 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.

3.2.1 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 [▶ 31]-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 der Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurück gesendet.

VAR_INPUT

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

CLEAR : 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.

VAR_OUTPUT

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

VALID : 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 : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde.

PORT : Enthält die Portnummer [▶ 108] des ADS-Quellgerätes von dem der ADS-Befehl gesendet wurde.

INVOKEID : Enthält ein Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

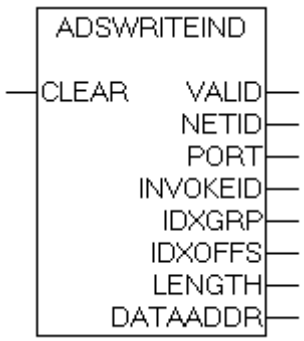
IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

LENGTH : Enthält die Anzahl der zu lesenden Daten in Bytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.2 ADSWRITEIND



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 [▶ 32]-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.

VAR_INPUT

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

CLEAR : 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.

VAR_OUTPUT

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

VALID : 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 : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde.

PORT : Enthält die Portnummer [▶ 108] des ADS-Quellgerätes von dem der ADS-Befehl gesendet wurde.

INVOKEID : Enthält ein Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

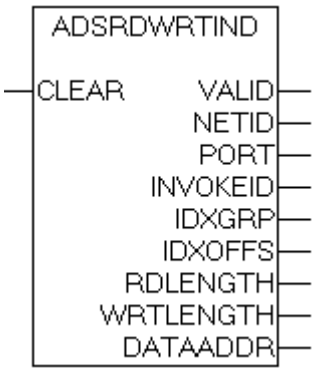
LENGTH : Enthält die Länge der geschriebenen Daten in Bytes.

DATAADDR : Enthält die Adresse des Datenpuffers in dem sich die geschriebenen Daten befinden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.3 ADSRDWRTIND



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 [▶ 33]-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 der Zuordnung der Antworten zu den Anfragen und wird ebenfalls als Parameter an das Quellgerät zurück gesendet.

VAR_INPUT

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

CLEAR : 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.

VAR_OUTPUT

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

VALID : 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 : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Quellgerätes enthält, von dem der ADS-Befehl gesendet wurde.

PORT : Enthält die Portnummer [▶ 108] des ADS-Quellgerätes von dem der ADS-Befehl gesendet wurde.

INVOKEID : Enthält ein Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.

IDXGRP : Enthält die Index-Gruppennummer (32bit, unsigned) des angeforderten ADS-Dienstes.

IDXOFFS : Enthält die Index-Offsetnummer (32bit, unsigned) des angeforderten ADS-Dienstes.

RDLENGTH : Enthält die Länge der zu lesenden Daten in Bytes.

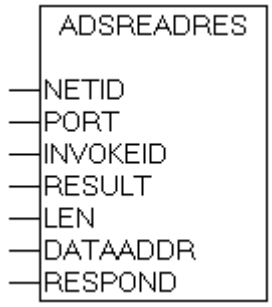
WRTLENGTH : Enthält die Länge der geschriebenen Daten in Bytes.

DATAADDR : Enthält die Adresse des Datenpuffers in dem sich die geschriebenen Daten befinden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.4 ADSREADRES



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 der Zuordnung der Antworten zu den Anfragen und wird von dem Ausgang des ADSREADIND [▶ 27]-Funktionsbaustein übernommen. Über den RESULT-Parameter kann ein Fehlercode an das ADS-Quellgerät zurückgegeben werden.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Quellgerätes enthält, an den der ADS-Befehl gesendet werden soll.

PORT : Enthält die Portnummer [▶ 108] des ADS-Quellgerätes an den die Antwort gesendet werden soll

INVOKEID : Enthält ein Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.

RESULT : Enthält die Fehlercode, der an das Quellgerät gesendet werden soll

LEN : Enthält die Anzahl der zu lesenden Daten in Bytes.

DATAADDR : Enthält die Adresse des Datenpuffers, der gelesen werden sollte.

RESPOND : Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

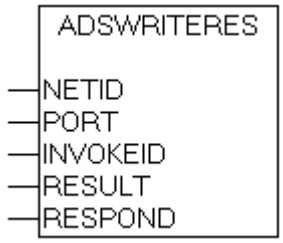
VAR_OUTPUT

```
(*keine*)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.5 ADSWRITERES



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 [▶ 27]-Funktionsbausteins übernommen. Über den RESULT-Parameter kann ein Fehlercode an das ADS-Quellgerät zurückgegeben werden.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Quellgerätes enthält, an den der ADS-Befehl gesendet werden soll.

PORT : Enthält die Portnummer [▶ 108] des ADS-Quellgerätes an den der ADS-Befehl gesendet werden soll

INVOKEID : Enthält ein Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.

RESULT : Enthält den Fehlercode, der an das Quellgerät gesendet werden soll

RESPOND : Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

VAR_OUTPUT

```
(*none*)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.6 ADSRDWRTRES

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 [▶ 30]-Funktionsbausteins übernommen. Über den RESULT-Parameter kann ein Fehlercode an das ADS-Quellgerät zurückgegeben werden.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Quellgerätes enthält, an den der ADS-Befehl gesendet werden soll.

PORT : Enthält die Portnummer [▶ 108] des ADS-Quellgerätes an den der ADS-Befehl gesendet werden soll.

INVOKEID : Enthält ein Handle des Befehls, der gesendet wurde. Die InvokeId wird von dem Quellgerät festgelegt und dient der Identifizierung der Befehle.

RESULT : Enthält den Fehlercode, der an das Quellgerät gesendet werden soll.

LEN : Länge der gelesenen Daten in Byte.

DATAADDR : Adresse des Datenpuffers in dem sich die gelesenen Daten befinden

RESPOND : Über eine positive Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

VAR_OUTPUT

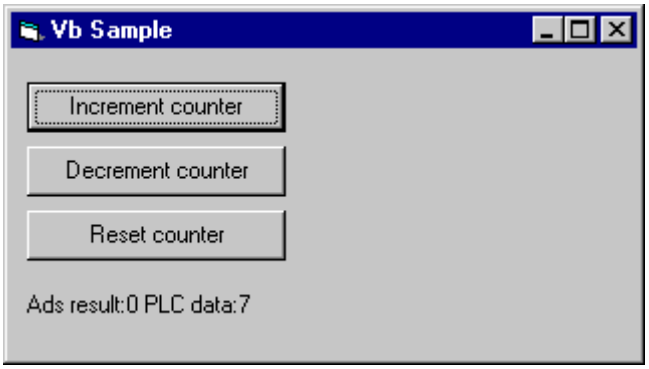
```
(*keine*)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.2.7 Beispiel 1: Erweiterte ADS-Funktionsbausteine

In der Beispielapplikation werden von einer VB-Applikation READ-Requests an die SPS-Task gesendet um eine SPS-Zählervariable zu inkrementieren/dekrementieren oder zurücksetzen. Beim Erfolg wird der Wert der Zählervariablen an die VB-Applikation zurück gesendet und auf der Form Ausgegeben. Um mit der SPS-Task zu kommunizieren benutzt die VB-Applikation das ActiveX Control: AdsOcx.



Hier können Sie die kompletten Sourcen zu der Beispielapplikation entpacken: https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828002187.exe

Die VB-Applikation

In der *Form_Load*-Routine wird eine Verbindung zu der SPS-Task aufgebaut (Port 802 auf dem lokalen Rechner). 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;

Damit die Requests an die SPS-Task weiter geleitet werden, muss in dem Index-Group-Parameter das höchstwertige Bit gesetzt werden. Der Index-Offset-Parameter ist Null.

```
Option Explicit

Dim tmpData(1) As Integer
Dim AdsResult    As Integer

Private Sub Command1_Click()

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

End Sub

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

End Sub

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

End Sub

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

Das SPS Programm

Die Requests werden in der SPS-Task von einer Instanz des ADSREADIND [▶ 27]-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 [▶ 31]-Funktionsbausteins an den Aufrufer mit dem aktuellen Wert der SPS-Variablen zurückgesendet. Im Fehlerfall eine entsprechende Fehlermeldung. Zum Schluss werden die Flags CLEAR und RESPOND rückgesetzt um weitere Indications verarbeiten zu können.

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

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

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

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

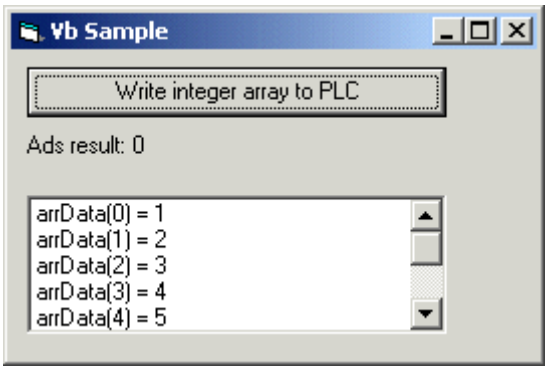
```
fbReadRes( RESPOND := FALSE );(* Reset response function block *)
fbReadInd( CLEAR := FALSE );(* Trigger indication function block *)
IF fbReadInd.VALID THEN(* Check for new indication *)

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

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


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

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

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

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

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

Hier können Sie die kompletten Sourcen zu der Beispielapplikation entpacken: https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828002187.exe

3.2.8 Beispiel 2: Erweiterte ADS-Funktionsbausteine

Die VB-Applikation sendet ein Array mit Integer-Werten zu einer SPS-Task. Die zuletzt gesendeten Werte werden einer Liste hinzugefügt und in der SPS in eine entsprechende Array-Variable kopiert. Die VB-Applikation benutzt einen WRITE-Request um die Daten zu senden. In der SPS wird der ADSWRITEIND-Funktionsbaustein benutzt um die Daten zu empfangen und ADSWRITERES-Funkstionbaustein um den WRITE-Request zu quittieren. Um mit der SPS-Task zu kommunizieren benutzt die VB-Applikation das ActiveX Control: AdsOcx.



Hier können Sie die kompletten Sourcen zu der Beispielapplikation entpacken: https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828002187.exe

Die VB-Applikation

In der *Form_Load*-Routine wird eine Verbindung zu der SPS-Task aufgebaut (Port 802 auf dem lokalen Rechner). Der gewünschte Dienst 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.

```
Option Explicit

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

Private Sub cmdWrite_Click()
    Call List1.Clear

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

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

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

End Sub

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

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

Das SPS Programm

Die Requests werden in der SPS-Task von einer Instanz des ADSWRITEIND [▶ 28]-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 [▶ 32]-Funktionsbausteins an den Aufrufer (eventuell mit einem Fehlercode) zurückgesendet. Zum Schluss 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.

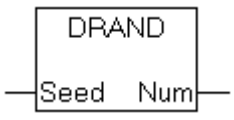
```
PROGRAM MAIN
VAR
    fbWriteInd  : ADSWRITEIND;
    fbWriteRes  : ADSWRITERES;

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

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

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

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

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

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

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

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

Hier können Sie die kompletten Sourcen zu der Beispielapplikation entpacken: https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828002187.exe

3.3 Generelle Funktionsbausteine

3.3.1 DRAND



Funktionsbausteine werden gemäß IEC61131-3 instanziiert und dann innerhalb des SPS-Programms über diesen Instanznamen aufgerufen beziehungsweise angesprochen. Der Funktionsbaustein DRAND erlaubt die Erzeugung einer (Pseudo-) Zufallszahl des Typs LREAL.

VAR_INPUT

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

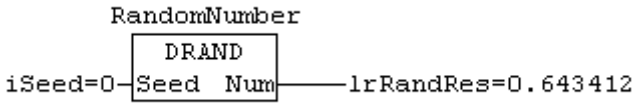
Seed : Initialwert zur Festlegung der Zufallszahlenreihe.

VAR_OUTPUT

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

Num : 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:

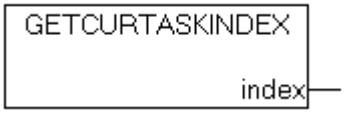


Im Beispiel wird der LREAL-Wert 0.643412 erzeugt und zurückgegeben. Mit dem Eingangsparameter "Seed" kann der Initialwert der Reihe beeinflusst werden. Wenn beispielsweise eine deterministisch reproduzierbare Zufallszahlenfolge in verschiedenen Sitzungen erreicht werden soll, muss ein identischer "Seed"-Wert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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.3.2 GETCURTASKINDEX



Funktionsbausteine werden gemäß IEC61131-3 instanziiert und dann innerhalb des SPS-Programms über diesen Instanznamen aufgerufen beziehungsweise angesprochen.

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

VAR_INPUT

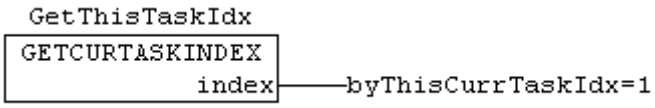
```
(*keine*)
```

VAR_OUTPUT

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

index : Gibt den aktuellen Taskindex der aufrufenden Task zurück (1..4).

Beispiel für den Aufruf des Bausteins in FBD:

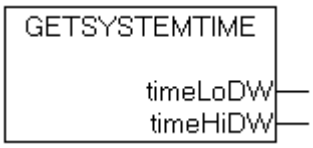


Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.4 Zeit-Funktionsbausteine

3.4.1 GETSYSTEMTIME



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.

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.

VAR_INPUT

```
(*keine*)
```

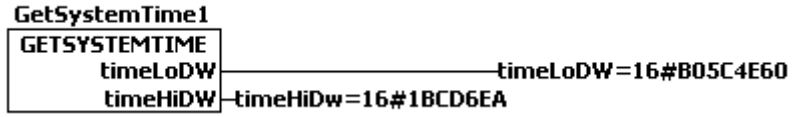
VAR_OUTPUT

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

timeLoDW : Enthält die niederwertigeren 4 Byte des Zeitstempels.

timeHiDW : Enthält die höherwertigeren 4 Byte des Zeitstempels.

Beispiel für den Aufruf des Bausteins in FBD:

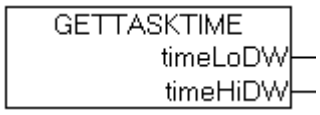


Das Beispiel zeigt den Aufruf des Bausteins über die Instanz ‚GetSystemTime1' und liefert den 64-bit, ganzzahligen Wert (hex), 1BCD6EAB05C4E60 als Zeitstempel.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.4.2 GETTASKTIME



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.

VAR_INPUT

```
(*keine*)
```

VAR_OUTPUT

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

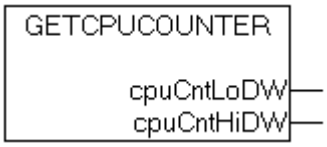
timeLoDW : Enthält die niederwertigeren 4 Byte des Zeitstempels.

timeHiDW : Enthält die höherwertigeren 4 Byte des Zeitstempels.

Voraussetzungen

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

3.4.3 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 100ns-Ticks umgerechnet ausgegeben wird. Der Zählwert wird, auf 100ns genau, bei jedem Aufruf durch das SPS System aufgefrischt und kann z.B. für Timing-Aufgaben eingesetzt werden. Eine Einheit entspricht 100ns. 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.

VAR_INPUT

```
(*keine*)
```

VAR_OUTPUT

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

cpuCntLoDW : Enthält die niederwertigeren 4 Byte des Zählwertes.

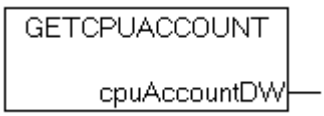
cpuCntHiDW : Enthält die höherwertigeren 4 Byte des Zählwertes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.4.4 GETCPUACCOUNT

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 100ns genau, bei jedem Aufruf der SPS-Task aufgefrischt und kann z.B. für Timing-Aufgaben eingesetzt werden. Eine Einheit entspricht 100ns.

VAR_INPUT

```
(*none*)
```

VAR_OUTPUT

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

cpuAccountDW: Der aktuelle Wert des Tickers der SPS-Task;

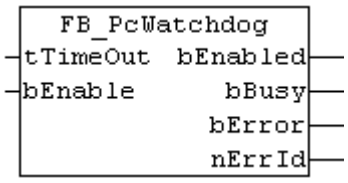
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib

3.5 Watchdog-Funktionsbausteine

3.5.1 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 1s und maximal 255s sein. Aktiviert wird der Watchdog über bEnable = TRUE und tTimeOut >= 1s.

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.

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

VAR_INPUT

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

tTimeOut: Watchdogzeit, nach deren Ablauf ein Neustart durchgeführt wird.

bEnable: Aktivieren bzw. Deaktivieren des Watchdogs.

VAR_OUTPUT

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

bEnabled : TRUE: Watchdog aktiviert / FALSE: Watchdog deaktiviert.

bBusy : Dieser Ausgang bleibt solange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt.

bError : 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 : Enthält den befehlsspezifischen 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:

```
VAR

    fbPcWatchDog  : FB_PcWatchdog;

    tWDTime       :
TIME := T#10s;

    bEnableWD     : BOOL;

    bWDActive     : BOOL;

END_VAR
```

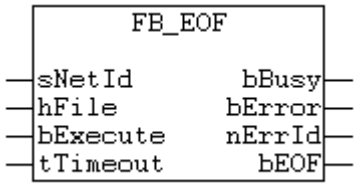
```
IF bEnableWD OR bWDActive THEN

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

    bWDActive := fbPcWatchDog.bEnabled;

END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.9.0 ab Build 1004	PC mit Mainboards: IP-4GVI63, CB1050, CB2050, CB3050, CB1051, CB2051, CB3051	TcSystem.Lib

3.6 Datei-Funktionsbausteine

3.6.1 FB_EOF



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

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

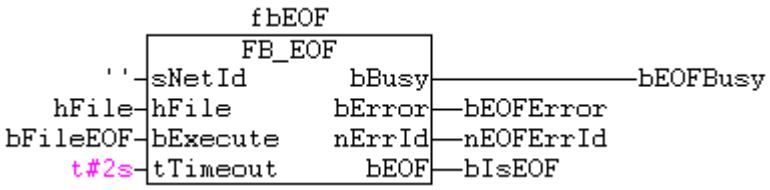
bEOF: Ist diese Variable gesetzt, dann wurde das Dateiende erreicht.

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

Beispiel für den Aufruf des Bausteins in FBD:

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

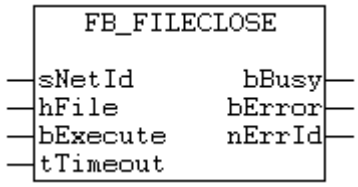
```
    bFileEOF    : BOOL;
    bEOFBusy    : BOOL;
    bEOFError   : BOOL;
    nEOFErrorId : UDINT;
    bIsEOF      : BOOL;
END_VAR
```



Voraussetzungen

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

3.6.2 FB_FileClose



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

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Handle der Datei, die geschlossen werden soll.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

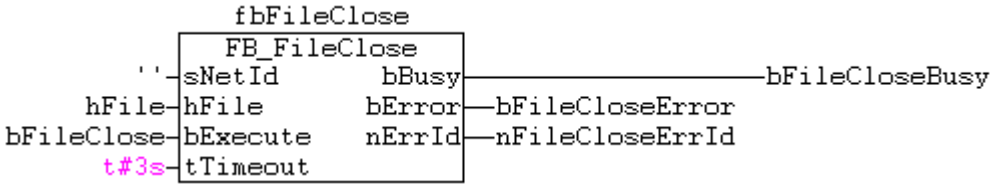
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

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

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
```

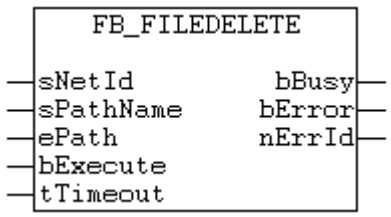


Hier wird die Datei, welche mit dem (durch "FB_FileOpen) File-Handle verknüpft ist, wieder geschlossen.

Voraussetzungen

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

3.6.3 FB_FileDelete



Der Funktionsbaustein löscht eine Datei auf dem Datenträger.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sPathName : Der Dateiname [▶ 109] mit dem gesamten Pfad.

ePath : Über diesen Eingang kann ein TwinCAT - Systempfad [▶ 110] auf dem Zielgerät zum Öffnen der Datei angewählt werden.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

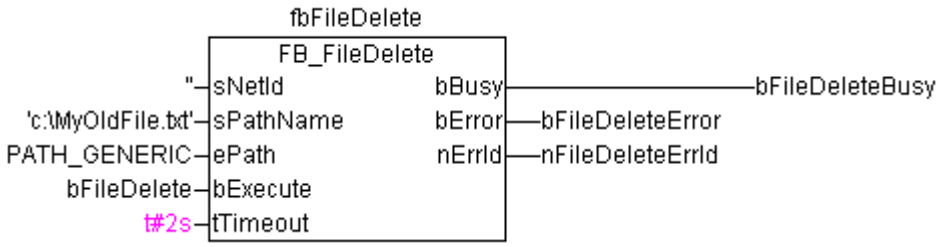
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

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

Beispiel für den Aufruf des Bausteins in FBD:

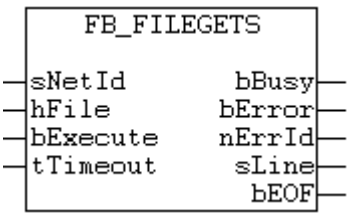
```
PROGRAM Test
VAR
    fbFileDelete    : FB_FileDelete;
    bFileDelete     : BOOL;
    bFileDeleteBusy : BOOL;
    bFileDeleteError: BOOL;
    nFileDeleteErrId: UDINT;
END_VAR
```



Voraussetzungen

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

3.6.4 FB_FileGets



Der Funktionsbaustein 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.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

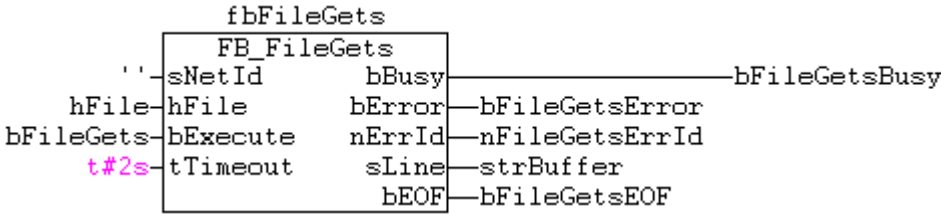
sLine : Der gelesene String [▶ 109].

bEOF : Ist diese Variable gesetzt, dann wurde das Dateiende erreicht und keine weiteren Datenbytes konnten gelesen werden (cbRead=0). Dieser Ausgang wird nicht gesetzt wenn noch einige Datenbytes gelesen werden konnten (cbRead>0).

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

Beispiel für den Aufruf des Bausteins in FBD:

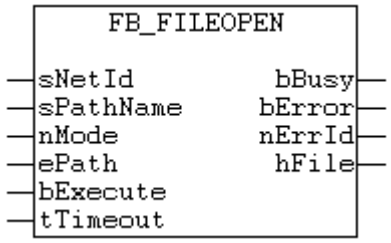
```
PROGRAM Test
VAR
    fbFileGets      : FB_FileGets;
    hFile           : UINT;
    bFileGets       : BOOL;
    bFileGetsBusy   : BOOL;
    bFileGetsError  : BOOL;
    nFileGetsErrorId: UDINT;
    strBuffer       : STRING;
    bFileGetsEOF    :BOOL;
END_VAR
```



Voraussetzungen

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

3.6.5 FB_FileOpen



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

VAR_INPUT

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

sNetId :Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sPathName : Enthält den Pfad- und Dateinamen [▶ 109] der zu öffnenden Datei.

Der Pfad kann nur auf das lokale File System des Rechners zeigen. Das bedeutet, Netzwerkpfade können hier nicht angegeben werden!

nMode : Enthält den 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, andernfalls 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

ePath : Über diesen Eingang kann ein TwinCAT - Systempfad [▶ 110] auf dem Zielgerät zum öffnen der Datei angewählt werden.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

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

VAR_OUTPUT

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

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

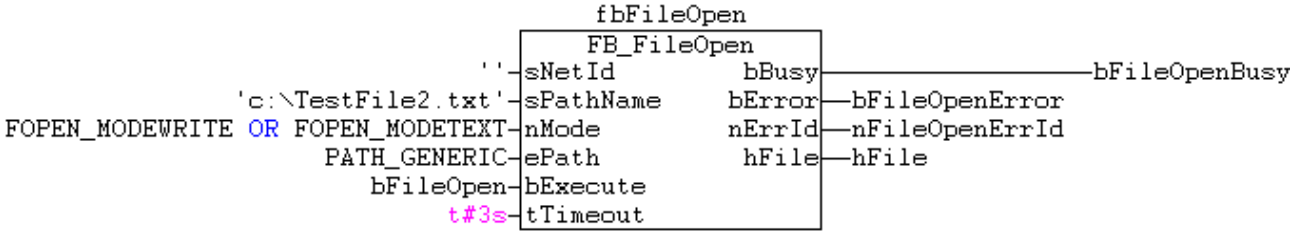
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

hFile: 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.

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.

Bei dem Öffnungsmodi 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:

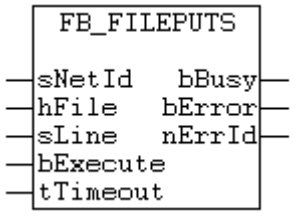
Textdatei Öffnungsmodi	Binärdatei Öffnungsmodi
FOPEN_MODEREAD OR FOPEN_MODETEXT	FOPEN_MODEREAD OR FOPEN_MODEBINARY
FOPEN_MODEWRITE OR FOPEN_MODETEXT	FOPEN_MODEWRITE OR FOPEN_MODEBINARY
FOPEN_MODEAPPEND OR FOPEN_MODETEXT	FOPEN_MODEAPPEND OR FOPEN_MODEBINARY
FOPEN_MODEREAD OR FOPEN_MODEPLUS OR FOPEN_MODETEXT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

Voraussetzungen

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

3.6.6 FB_FilePuts



Der Funktionsbaustein 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.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

sLine : Der String [▶ 109] der in die Datei geschrieben werden soll.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

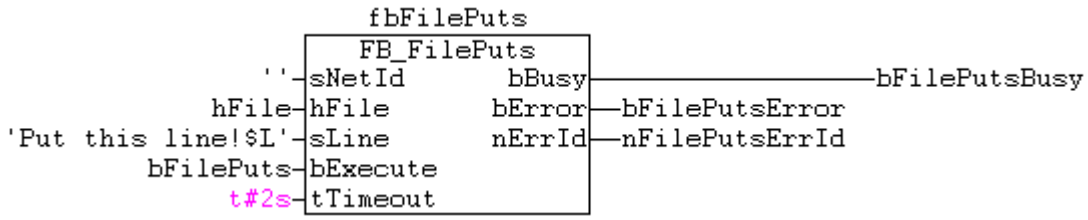
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

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

Beispiel für den Aufruf des Bausteins in FBD:

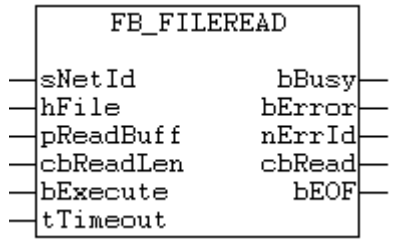
```
PROGRAM Test
VAR
    fbFilePuts      : FB_FilePuts;
    hFile           : UINT;
    bFilePuts       : BOOL;
    bFilePutsBusy   : BOOL;
    bFilePutsError  : BOOL;
    nFilePutsErrorId: UDINT;
END_VAR
```



Voraussetzungen

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

3.6.7 FB_FileRead



Mit diesem Funktionsbaustein kann der Inhalt einer bereits geöffneten Datei ausgelesen werden. Die Datei muss vor einem Lesezugriff im entsprechenden Modus geöffnet worden sein.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

pReadBuff : Enthält die 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 : Enthält die Anzahl der zu lesenden Bytes.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

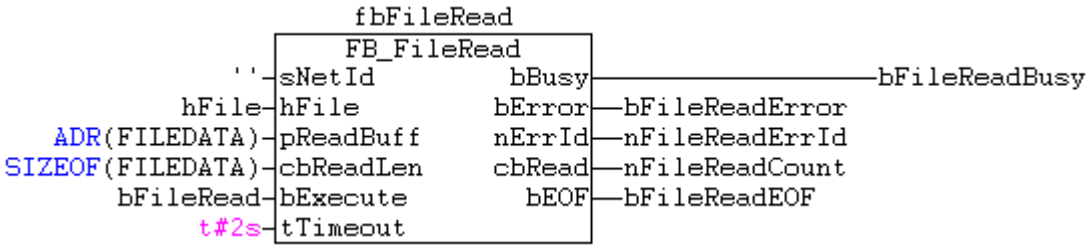
cbRead : Enthält die Anzahl der aktuell gelesenen Bytes.

bEOF : Ist dieser Ausgang gesetzt, dann wurde das Dateiende erreicht und keine weiteren Datenbytes konnten gelesen werden (cbRead=0). Dieser Ausgang wird nicht gesetzt wenn noch einige Datenbytes gelesen werden konnten (cbRead>0).

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

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
```

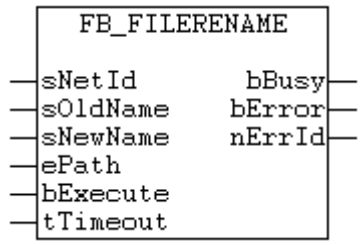


Nach steigender Flanke von "bExecute" und erfolgreicher Ausführung des Lese-Befehls, befinden sich in "FILEDATA" die aktuell gelesenen Bytes der Datei. Wie viele Bytes beim jeweils letzten Lesevorgang tatsächlich gelesen wurden, kann anhand der Parameters "cbRead" ermittelt werden.

Voraussetzungen

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

3.6.8 FB_FileRename



Mit diesem Funktionsbaustein kann eine Datei umbenannt werden.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sOldName : Der alte Dateiname [▶ 109].

sNewName : Der neue Dateiname.

ePath : Über diesen Eingang kann ein TwinCAT - Systempfad [▶ 110] auf dem Zielgerät zum öffnen der Datei angewählt werden.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

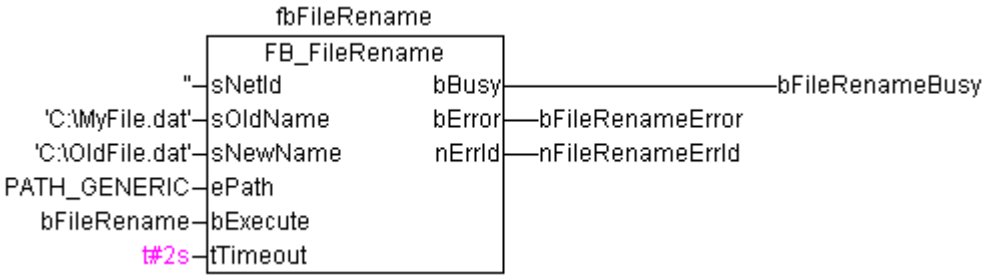
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

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

Beispiel für den Aufruf des Bausteins in FBD:

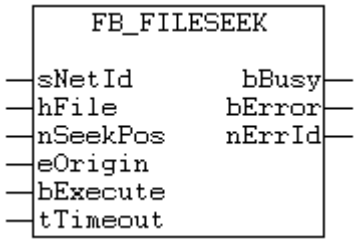
```
PROGRAM Test
VAR
    fbFileRename    : FB_FileRename;
    bFileRename     : BOOL;
    bFileRenameBusy : BOOL;
    bFileRenameError: BOOL;
    nFileRenameErrId: UDINT;
END_VAR
```



Voraussetzungen

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

3.6.9 FB_FileSeek



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

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

nSeekPos : Neue Position de Dateizeigers.

eOrigin : Relative Position [▶ 110] zu der der Dateizeiger bewegt werden soll.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

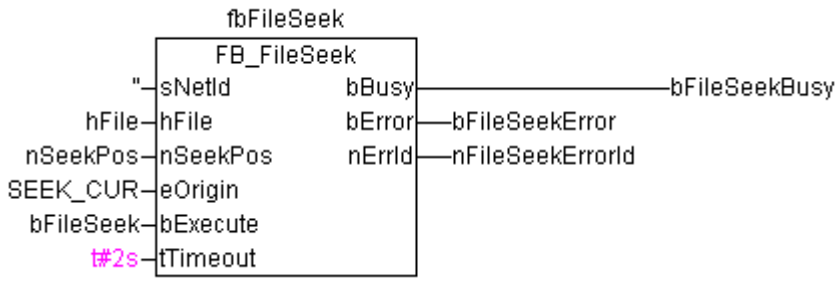
bError: Sollte ein ADS-Fehler bei der Übertragung des Kommandos erfolgen, dann wird dieser Ausgang gesetzt, nachdem der bBusy-Ausgang zurückgesetzt wurde.

nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

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

Beispiel für den Aufruf des Bausteins in FBD:

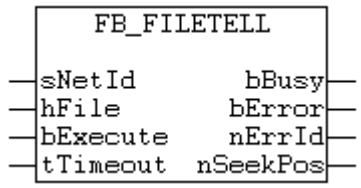
```
PROGRAM Test
VAR
    fbFileSeek      : FB_FileSeek;
    hFile           : UINT;
    nSeekPos        : DINT;
    bFileSeek       : BOOL;
    bFileSeekBusy   : BOOL;
    bFileSeekError  : BOOL;
    nFileSeekErrorId: UDINT;
END_VAR
```



Voraussetzungen

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

3.6.10 FB_FileTell



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

Bitte beachten Sie, dass bei Dateien, die in dem Modi "Anhängen ans Ende der Datei" geöffnet wurden die aktuelle Position durch die letzte I/O-Operation bestimmt wird, 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 keine I/O-Operation bis jetzt durchgeführt wurde und die Datei im Modi zum Anhängen geöffnet wurde dann steht der Dateizeiger am Dateianfang.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

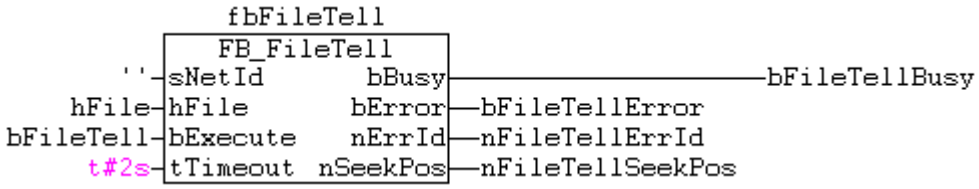
nErrId: Liefert bei einem gesetzten bError-Ausgang die ADS-Fehlernummer.

nSeekPos : Liefert die aktuelle Position des Dateizeigers.

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

Beispiel für den Aufruf des Bausteins in FBD:

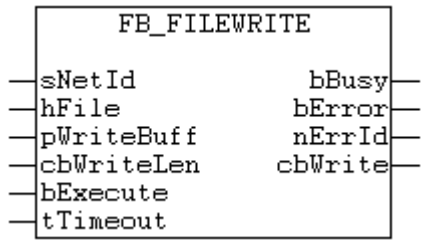
```
PROGRAM Test
VAR
    fbFileTell      : FB_FileTell;
    hFile           : UINT;
    bFileTell       : BOOL;
    bFileTellBusy   : BOOL;
    bFileTellError  : BOOL;
    nFileTellErrorId: UDINT;
    nFileTellSeekPos: DINT;
END_VAR
```



Voraussetzungen

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

3.6.11 FB_FileWrite



Der Funktionsbaustein 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.

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hFile : Datei-Handle.

pWriteBuff : Enthält die 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 : Enthält die Anzahl der zu schreibenden Bytes.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

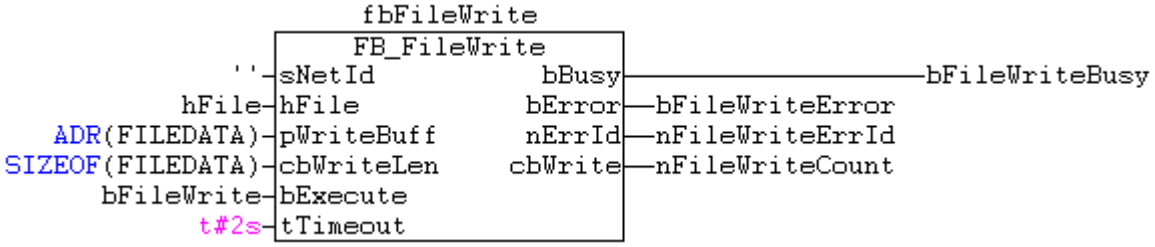
nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

cbWrite : 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.

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

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
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.12 Beispiel: Dateizugriff aus der SPS

Systemvoraussetzungen:

- TwinCAT 2.9;

In diesem Beispiel wird die Anwendung der SPS-Funktionsbausteine für den Dateizugriff aus der TcSystem.Lib vorgestellt. Mit Hilfe der vorhandenen Funktionsbausteine wurde ein neuer Funktionsbaustein FB_FileCopy realisiert. Mit diesem Baustein können Binärdateien z.B. auf dem lokalen TwinCAT PC oder von einem Remote-TwinCAT PC auf den lokalen TwinCAT PC kopiert werden. Mit den Funktionsbausteinen 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;

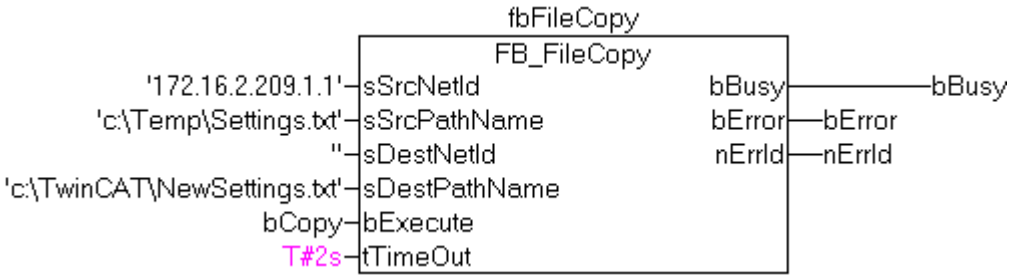
d) Überprüfen, ob das Ende der Quell-Datei erreicht wurde. Wenn nicht dann b) und c) wiederholen. Wenn ja, dann zu e) springen;

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. Die kompletten Sourcen zu dem Beispielprojekt können hier entpackt werden: https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828003595.exe.

Das SPS-Programm:

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



Im Beispiel wird die *Settings.txt*-Datei aus dem *Temp*-Ordner von einem Remote-TwinCAT-PC mit der Netzwerkadresse: "172.16.2.209.1.1" in den *TwinCAT*-Ordner auf dem lokalen TwinCAT PC kopiert.

Der FB_FileCopy-Funktionsbaustein

Interface:

```
FUNCTION_BLOCK FB_FileCopy
VAR_INPUT
    sSrcNetId           : T_AmsNetId;(* TwinCAT network address of the source file *)
    sSrcPathName        : T_MaxString;(* Source file path and name *)
    sDestNetId          : T_AmsNetId;(* TwinCAT network address of the destination file *)
    sDestPathName       : T_MaxString; (* Destination file path and name *)
    bExecute            : BOOL; (* Rising edge start fb execution *)
    tTimeOut            : TIME := DEFAULT_ADS_TIMEOUT;(* Max. ADS timeout time *)
END_VAR
VAR_OUTPUT
    bBusy        :BOOL;
```

```
(* TRUE => File copy execution in progress, FALSE => File copy execution idle *)
 bError          :BOOL;(* TRUE => Error, FALSE => No error *)
 nErrId          :UDINT;(* Error code *)
END_VAR
VAR
    fbFileOpen          :FB_FileOpen;
    fbFileClose         :FB_FileClose;
    fbFileRead          :FB_FileRead;
    fbFileWrite         :FB_FileWrite;

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

    Step            :DWORD;
    RisingEdge      :R_TRIG;

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

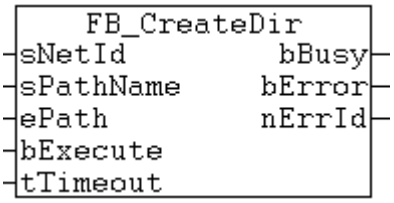
Implementierung:

```
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
```

Die kompletten Sourcen zu dem Beispielprojekt können hier entpackt werden: https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828003595.exe.

3.6.13 FB_CreateDir



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

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sPathName : Der Name des neuen Verzeichnisses als String [▶ 109]. Es kann nur ein neuer Verzeichniss beim einmaligen Aufruf des Bausteins erstellt werden.

ePath : Über diesen Eingang kann ein TwinCAT - Systempfad [▶ 110] für das neue Verzeichnis auf dem Zielgerät angewählt werden.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

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

Beispiel in ST:

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

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

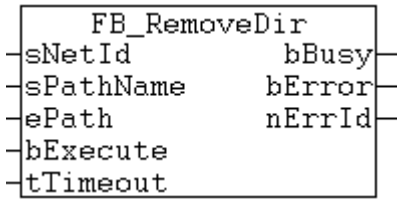
```
PROGRAM MAIN
VAR
    sFolderName : STRING := 'PRJDATA'; (* folder name *)
    bBootFolder : BOOL;

    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
TwinCAT v2.10.0 Build > 1310 (CE image v2.17d or higher)	PC oder CX (x86, ARM)	TcSystem.Lib

3.6.14 FB_RemoveDir



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

VAR_INPUT

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

sNetId : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

sPathName : Der zu löschende Verzeichnissname [▶ 109]. Es kann nur ein Verzeichniss beim einmaligen Aufruf des Funktionsbausteins gelöscht werden. Die letzte Komponente von sPathName muss den zu löschenden Verzeichnissnamen beinhalten.

ePath : Über diesen Eingang kann ein TwinCAT - Systempfad [▶ 110] zum Löschen des Verzeichnis auf dem Zielgerät angewählt werden.

bExecute : Durch eine steigende Flanke an diesem Eingang wird der Funktionsbaustein aktiviert.

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

VAR_OUTPUT

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

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

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

nErrId: Liefert bei einem gesetzten *bError*-Ausgang die ADS-Fehlernummer.

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

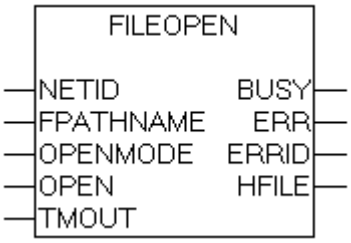
Beispiel in ST: Siehe in der Beschreibung von FB_CreateDir [▶ 65].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1310 (CE image v2.17d or higher)	PC oder CX (x86, ARM)	TcSystem.Lib

3.6.15 TwinCAT 2.7 file function blocks

3.6.15.1 FILEOPEN



Mit diesem Funktionsbaustein kann eine Datei neu angelegt, bzw. eine bereits bestehende und geschlossene Datei zur weiteren Bearbeitung erneut geöffnet werden.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

FPATHNAME : Enthält den Pfad- und Dateinamen [▶ 109] der zu öffnenden Datei.

Der Pfad kann nur auf das lokale Filesystem des Rechners zeigen. Das bedeutet, Netzwerkpfade können hier nicht angegeben werden!

OPENMODE : Enthält den 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.

- FILE_OPENCREATE Erzeugen und Öffnen einer Datei (Bereits bestehende Dateien werden bei diesem Modus überschrieben!). FILE_OPENREAD Öffnen einer Datei für lesenden Zugriff;
- FILE_OPENWRITE Öffnen einer Datei zum Schreiben beginnt am Anfang der Datei.

OPEN : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

BUSY : Dieser Ausgang bleibt solange 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.

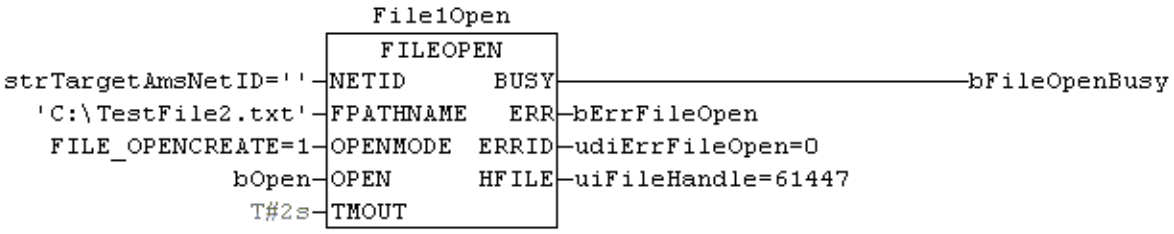
Es wird nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

HFILE : Enthält, nach erfolgreichem Öffnen, das erzeugte File-Handle für die Datei.

Beispiel für den Aufruf des Bausteins in FBD:



Hierbei soll die Datei "TestFile2.txt" im Root-Verzeichnis des Laufwerks "C:" angelegt bzw. Überschrieben werden. Im Beispiel ist ein ordnungsgemäßes Handle für die Datei erzeugt worden. Dieses Handle wird nun an die nachfolgend beschriebenen File-Funktionsbausteine als Kennung für die zu bearbeitende Datei übergeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.2 FILECLOSE

Mit diesem Funktionsbaustein kann die Datei geschlossen und damit in einen definierten Zustand zur weiteren Verarbeitung durch andere Programme versetzt werden.

VAR_INPUT

```
VAR_INPUT
    NETID   : T_AmsNetId;    (* ams net id *)
    HFILE   : UINT;          (* file handle obtained through 'FILEOPEN' *)
```

```
    CLOSE   : BOOL;         (* close control input *)
    TMOUT   : TIME;
END_VAR
```

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

HFILE : Wird mit dem durch FILEOPEN bereits erzeugten File-Handle belegt.

CLOSE : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

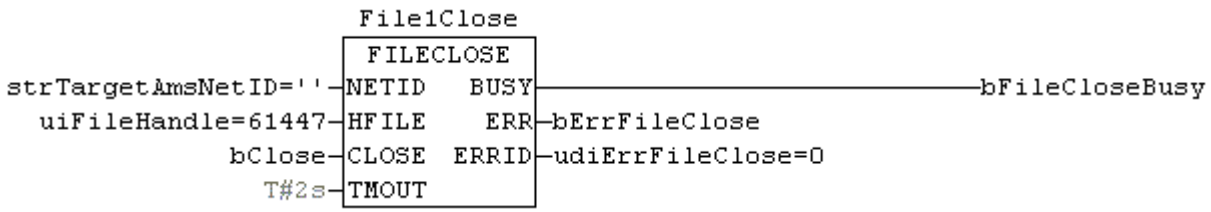
```
VAR_OUTPUT
    BUSY    : BOOL;
    ERR     : BOOL;
    ERRID   : UDINT;
END_VAR
```

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen 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:



Hier wird die Datei, welche mit dem (durch "File1Open" erzeugten) File-Handle verknüpft ist, wieder geschlossen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.3 FILEWRITE

Mit diesem Funktionsbaustein können Daten in eine Datei geschrieben werden. Die Datei muss für einen Schreibzugriff zuvor mit dem Modus "FILE_OPENCREATE" oder "FILE_OPENWRITE" geöffnet worden sein und zur weiteren Verarbeitung durch externe Programme wieder geschlossen werden.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

HFILE : Wird mit dem, durch FILEOPEN bereits erzeugten, File-Handle belegt.

BUFADDR : Enthält die 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.

COUNT : Enthält die Anzahl der zu schreibenden Bytes.

WRITE : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

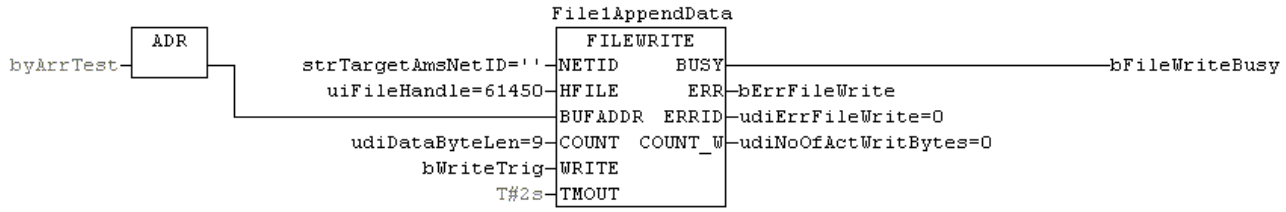
BUSY : Dieser Ausgang bleibt solange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der, an dem 'Timeout'-Eingang angelegten, Zeit. Während Busy = TRUE wird an den Eingängen kein neuer Befehl angenommen. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt. Ist der Fehlercode -1 (16#FFFF), so kann der Schreibvorgang nicht durchgeführt werden, weil z.B. die Datei nicht korrekt geöffnet wurde.

COUNT_W : Enthält die Anzahl der aktuell geschriebenen Bytes.

Beispiel für den Aufruf des Bausteins in FBD:

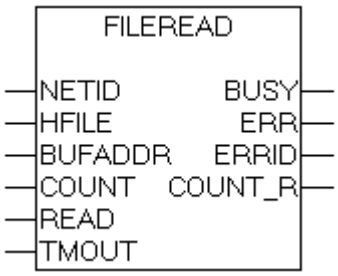


Im Beispiel werden, nach steigender Flanke von "bWriteTrig", 9 Byte des Arrays "byArrTest" an das Ende der Datei mit dem Handle "61450" geschrieben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.4 FILEREAD



Mit diesem Funktionsbaustein kann der Inhalt einer bereits geöffneten Datei ausgelesen werden. Die Datei muss vor einem Lesezugriff mit dem Modus "FILE_OPENREAD" geöffnet worden sein.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

HFILE : Wird mit dem, durch FILEOPEN bereits erzeugten, File-Handle belegt.

BUFADDR : Enthält die Adresse des Puffers, in den die Daten gelesen werden. Der Puffer kann eine Einzelvariable, ein Array oder eine Struktur sein, dessen Adresse mit dem ADR - Operator ermittelt werden kann.

COUNT : Enthält die Anzahl der zu lesenden Bytes.

READ : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

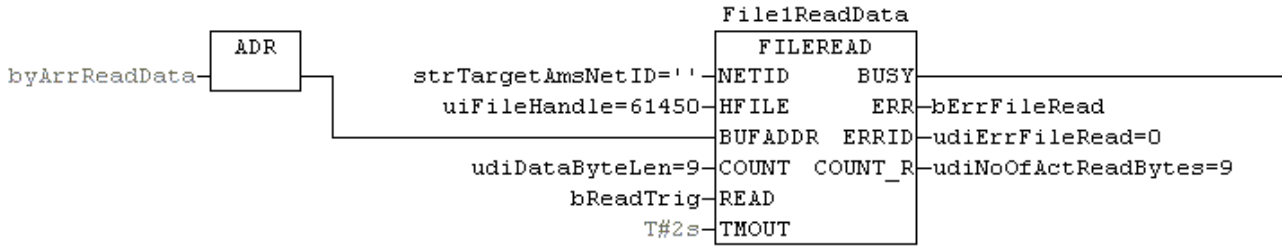
BUSY : Dieser Ausgang bleibt solange auf TRUE, bis der Baustein eine Befehlsanforderung ausführt, längstens aber für die Dauer der, an dem 'Timeout'-Eingang angelegten, Zeit. Während Busy = TRUE wird an den Eingängen kein neuer Befehl angenommen. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt. Ist der Fehlercode -1 (16#FFFF), so kann der Schreibvorgang nicht durchgeführt werden, weil z.B. die Datei nicht korrekt geöffnet wurde.

COUNT_R : Enthält die Anzahl der aktuell gelesenen Bytes.

Beispiel für den Aufruf des Bausteins in FBD:

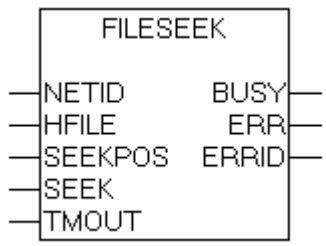


Nach steigender Flanke von "bReadTrig" und erfolgreicher Ausführung des Lese-Befehls, befinden sich in "byArrReadData" die aktuell gelesenen Bytes der Datei. Wie viele Bytes beim jeweils letzten Lesevorgang tatsächlich gelesen wurden, kann anhand der Parameters "COUNT_R" ermittelt werden. Im Beispiel wird angezeigt, dass neun Bytes gelesen wurden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.6.15.5 FILESEEK



Mit diesem Funktionsbaustein kann der Dateizeiger einer geöffneten Datei auf eine definierbare Position gesetzt werden.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung [▶ 107] des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

HFILE : Wird mit dem, durch FILEOPEN bereits erzeugten, File-Handle belegt.

SEEKPOS : Enthält die gewünschte (absolute) Zielposition des Dateizeigers. Die Konstanten FILE_SEEKEND und FILE_SEEKBEGIN können benutzt werden um den Dateizeiger an das Ende bzw. an den Anfang der Datei zu setzen.

SEEK : Durch eine steigende Flanke an diesem Eingang wird der ADS-Befehl ausgelöst.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

BUSY : Dieser Ausgang bleibt solange 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. Bitte beachten Sie, dass nicht die Ausführung des Dienstes, sondern nur dessen Annahme zeitlich überwacht wird.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

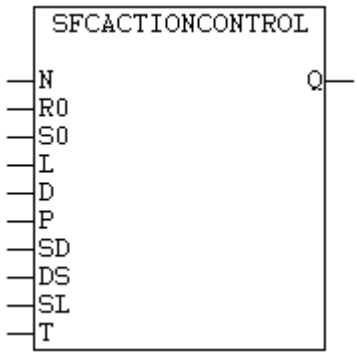
ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7 IEC-Schritte / SFC-Flags FB's

3.7.1 SFCActionControl

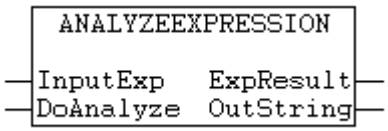


Dieser Funktionsbaustein wird benötigt um in SFC-Programmen/-Projekten die IEC-Schritte benutzen zu können. Im Projekt muss nur die Bibliothek mit dem FB eingebunden werden. Es werden aber keine Instanzen benötigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	IecSfc.Lib
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7.2 AnalyzeExpression

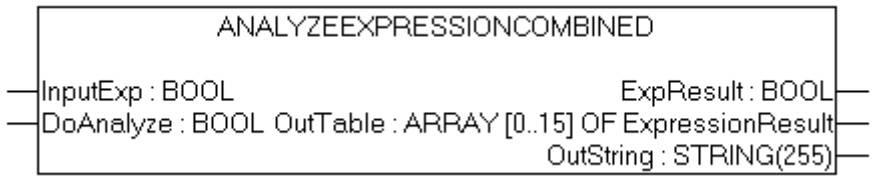


Der Funktionsbaustein wird in SPS-Projekten benötigt, die die SFC-Flags benutzen. Es werden keine Instanzen erzeugt. Im Projekt muss nur die entsprechende SPS-Bibliothek eingebunden werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Analyzation.Lib
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7.3 AnalyzeExpressionCombined

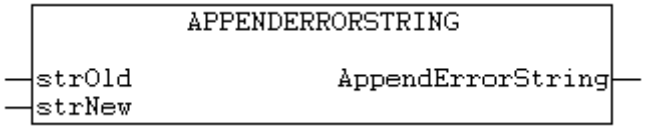


Der Funktionsbaustein wird in SPS-Projekten benötigt, die die SFC-Flags benutzen. Es werden keine Instanzen erzeugt. Im Projekt muss nur die entsprechende SPS-Bibliothek eingebunden werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.7.4 AppendErrorString



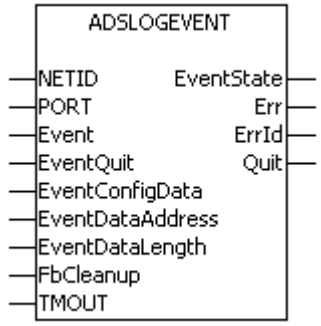
Die Funktion wird in SPS-Projekten benötigt, die die SFC-Flags benutzen. Die Funktion muss nicht im Projekt aufgerufen werden, es muss nur die entsprechende SPS-Bibliothek eingebunden werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Analyzation.Lib
TwinCAT v2.8.0 Build > 718	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.8 Eventlogger Bausteine

3.8.1 ADSLOGEVENT



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

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

PORT : Enthält die Portnummer des ADS-Gerätes. Der TwinCAT Eventlogger hat die Portnummer 110.

Event : Mit der steigenden Flanke wird das 'Kommen' des Events signalisiert. Mit der fallenden Flanke das 'Gehen' des Events.

EventQuit : Mit der steigenden Flanke wird das Event quittiert.

EventConfigData : Datenstruktur mit den Event-Parametern [▶ 112].

EventDataAddress : Adresse mit den Daten, die mit dem Event geschickt werden sollen.

EventDataLength : Länge der Daten, die mit dem Event geschickt werden sollen.

FbCleanup : Bei TRUE wird der Baustein komplett initialisiert.

TMOUT : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

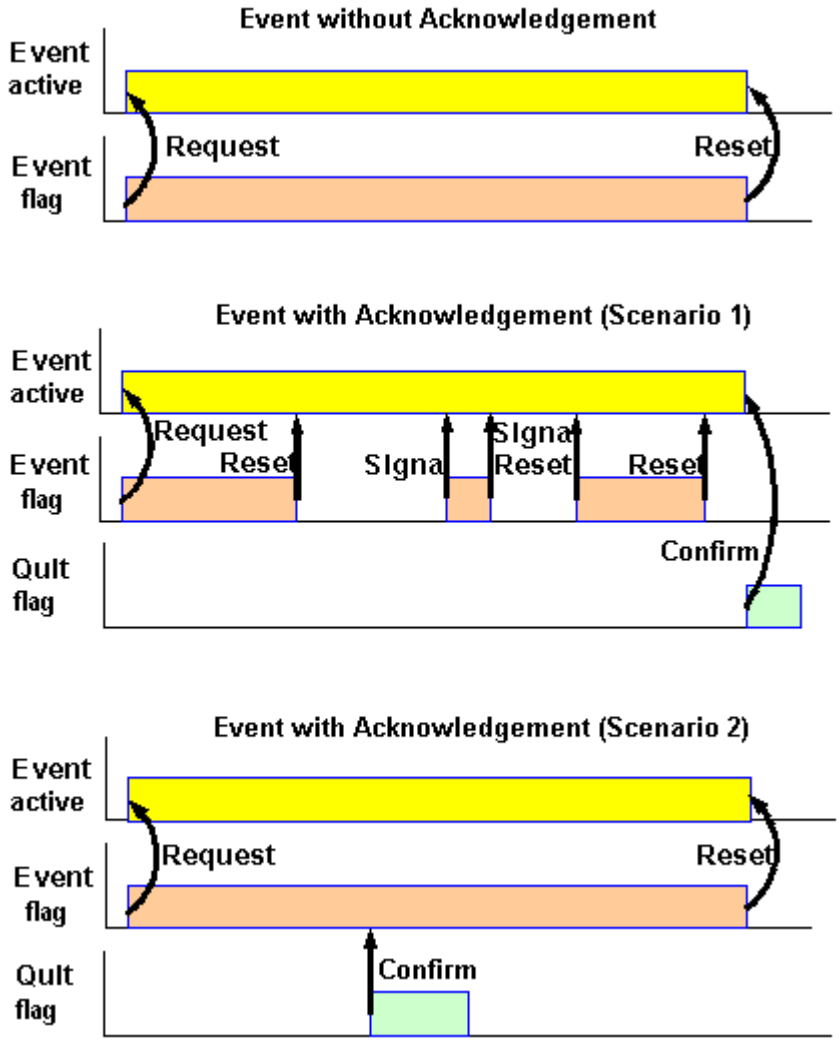
```
VAR_OUTPUT
    EventState : UDINT;
    Err        : BOOL;
    ErrId      : UDINT;
    Quit       : BOOL;
END_VAR
```

EventState : Zustand des Events.

ERR : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'ErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'Error' = TRUE und 'ErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

ERRID : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Quit : Quittiert das Event.



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 Eventaktivierung und Kommen der Quittierung ein Reset des Event kommt, heißt das nächste Kommen des Event Eingangs 'Signal'. Damit wird ein Request bei bereits aktiven Event gemeldet.

Schrittweiser Ablauf:

- **Konfigurieren eines Events:**

Struktur EventConfigData [▶ 112] 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 [▶ 115] sichtbar am Eventstate Ausgang.

Beispiel:

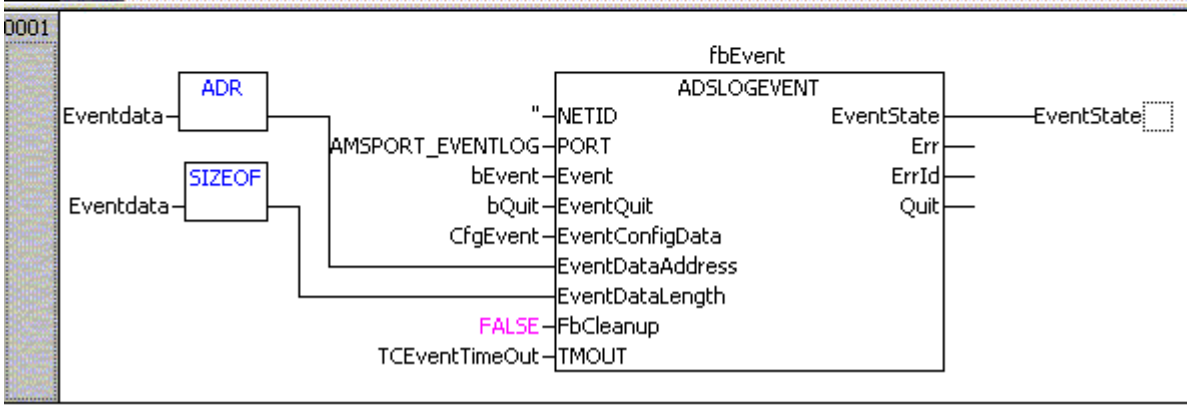
Konfiguration des Events:

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

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

Aufruf des FBs ADSLOGEVENT:

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



https://infosys.beckhoff.com/content/1031/tcplclibsystem/Resources/11828005003.zip

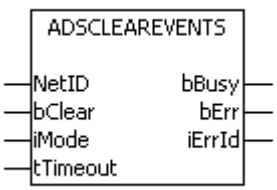
Dieses Sample enthält

- Eine 'Event Configuration.ecp' Datei, welche mit dem TcEventConfigurator geöffnet und aktiviert werden kann.
- Ein SPS Beispiel Programm

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCEvent.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

3.8.2 ADSCLEAREVENTS



Der Funktionsbaustein sendet und quittiert Meldungen zum TwinCAT Eventlogger.

VAR_INPUT

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

NETID : Ist ein String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

bClear : Mit der steigenden Flanke werden die Events gelöscht.

iMode : Mode zum Löschen der Events. Definiert im Enum E_TcEventClearModes [▶ 111].

tTimeout : Gibt die Zeit bis zum Abbruch der Funktion an.

VAR_OUTPUT

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

bBusy : Ist TRUE solange die Aktion ausgeführt wird. In dieser Zeit ist kein neues Kommando möglich.

bErr : Dieser Ausgang wird auf TRUE geschaltet, wenn bei der Ausführung eines Befehls ein Fehler aufgetreten ist. Der befehlsspezifische Fehlercode ist in 'iErrorId' enthalten. Wenn der Baustein ein Timeout-Fehler hat, so ist 'bErr' = TRUE und 'iErrorId' = 1861 (Hexadezimal 0x745). Wird durch das Ausführen eines Befehls an den Eingängen auf FALSE zurückgesetzt.

iErrId : Enthält den befehlsspezifischen Fehlercode des zuletzt ausgeführten Befehls. Wird durch das Ausführen eines Befehls an den Eingängen auf 0 zurückgesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4 Funktionen

4.1 Generelle Funktionen

4.1.1 F_SplitPathName

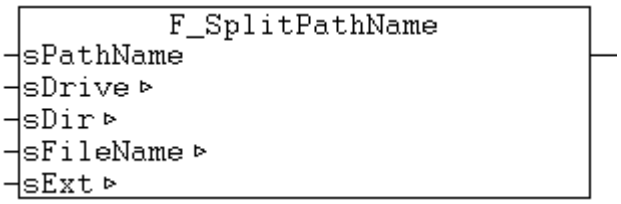


Abb. 1: F_SplitPathName

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

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

sPathName: Vollständiger Dateiname als String [▶ 109] in der Form: 'X:\DIR\SUBDIR\FILENAME.EXT'.

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

sDrive: Laufwerksbezeichner mit einem Doppelpunkt ('C:', 'A:' usw.).

sDir: Verzeichnisname inklusive dem führenden und abschließenden backslash ('\BC \INCLUDE\', '\SOURCE\' usw.).

sFileName: Dateiname.

sExt: Enthält den Punkt, der die Namenserweiterung einleitet ('.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;
```

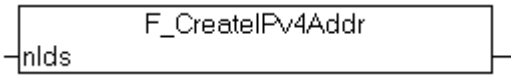
```
    sPathName   : T_MaxString := 'C:\TwinCAT\Plc\Project01\Data.txt';
    sDrive      : STRING(3);
    sDir        : T_MaxString;
    sFileName       : T_MaxString;
    sExt        : T_MaxString;
    bSuccess    : BOOL;
END_VAR
```

```
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
TwinCAT v2.10.0 Build > 1307	PC or CX (x86) CX (ARM)	TcSystem.Lib

4.1.2 F_CreateIPv4Addr



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

FUNCTION F_CreateIPv4Addr : T_IPv4Addr

T_IPv4Addr [▶ 109]

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

nIds: Byte-Array [▶ 109]. Jedes Byte entspricht einem Adressbyte der (IPv4) Internet Protocol Netzwerkadresse. Die Adressbytes haben die Netzwerk-Byte-Reihenfolge.

Beispiel für einen Aufruf in ST:

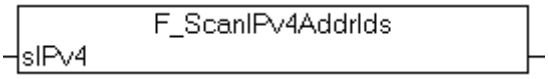
```
PROGRAM MAIN
VAR
    ids     : T_IPv4AddrArr := 172, 16, 7, 199;
    sIPv4   : T_IPv4Addr := '';
END_VAR
```

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

Voraussetzungen

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

4.1.3 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

T_IPv4AddrArr [▶ 109]

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

sIPv4: Internet Protocol Netzwerkadresse [▶ 109] als String. 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.

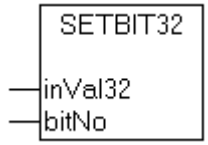
```
PROGRAM MAIN
VAR
    ids     : T_IPv4AddrArr;
    sIPv4   : T_IPv4Addr := '172.16.7.199';
END_VAR
```

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

Voraussetzungen

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

4.1.4 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

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

inVal32: Der zu verändernde 32-bit Wert;

bitNo: Die Nummer des zu setzenden Bits (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet;

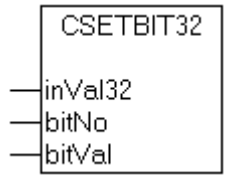
Beispiel für den Aufruf der Funktion in FBD:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.5 CSETBIT32



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

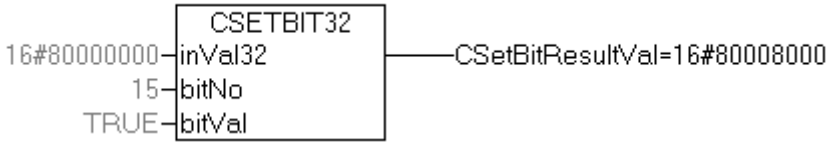
```
VAR_INPUT
    inVal32     : DWORD;
    bitNo       : SINT;
    bitVal      : BOOL;
END_VAR
```

inVal32: Ein 32-bit Wert;

bitNo: Die Nummer des Bits, der gesetzt bzw. zurückgesetzt werden soll (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet;

bitVal: Der 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
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.6 GETBIT32

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.

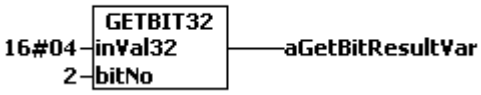
FUNCTION GETBIT32 : BOOL

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

inVal32: Der 32-bit Wert;

bitNo: Die Nummer des Bits, der gelesen werden soll (0-31). Diese Zahl wird vor der Ausführung intern modulo 32 verrechnet;

Beispiel für den Aufruf der Funktion in FBD:

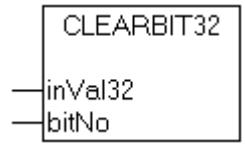


Hierbei wird das Bit 2 in dem Eingangswert ‚16#04' abgefragt und der boolschen Variablen *aGetBitResultVar* zugewiesen. Die Abprüfung ergibt in diesem Beispiel TRUE.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.7 CLEARBIT32



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.

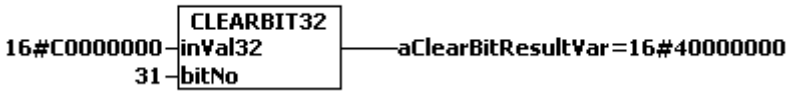
FUNCTION CLEARBIT32 : DWORD

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

inVal32: Der zu verändernde 32-bit Wert;

bitNo: Die 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:

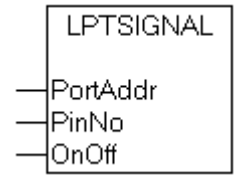


Hierbei wird das Bit 31 in dem Eingangswert ‚C0000000' zurückgesetzt. Es ergibt sich der Wert (hex) ‚40000000'.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.8 LPTSIGNAL



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

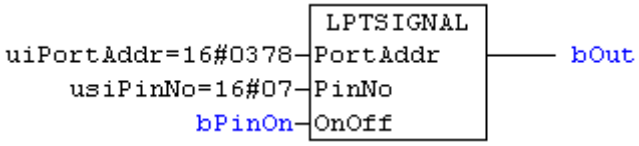
```
VAR_INPUT
    PortAddr    : UINT;
    PinNo       : INT;
    OnOff       : BOOL;
END_VAR
```

PortAddr: Adresse des Ports, welcher für die gewünschte LPT-Schnittstelle zur Verfügung steht.

PinNo: Enthält die Nummer des Pins (Pin 0 .. 7), der von der PLC beschrieben werden soll.

OnOff: Enthält den Zustand, der an den Pin herausgeschrieben werden soll.

Beispiel für den Aufruf der Funktion in FBD:

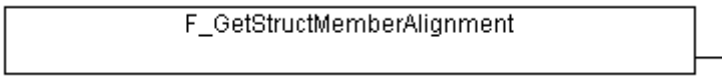


Im Beispiel wird Bit 7 des Ports 378 (hex) auf 1 gesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.1.9 F_GetStructMemberAlignment



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

FUNCTION F_GetStructMemberAlignment : BYTE

```
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

Folgende 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 Srukturelemente ändert sich die Anordung 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
```

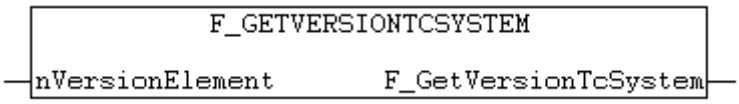
```
test5 : ST_TEST5;
```

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11 Build > 1553 TwinCAT v2.11 R2 Build > 2034	PC or CX (x86, ARM)	TcSystem.Lib

4.1.10 F_GetVersionTcSystem



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

FUNCTION F_GetVersionTcSystem : UINT

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

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

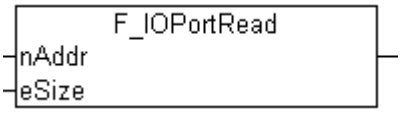
- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Voraussetzungen

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

4.2 I/O Portzugriff

4.2.1 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 [▶ 92] Funktion.

FUNCTION F_IOPortRead : DWORD

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

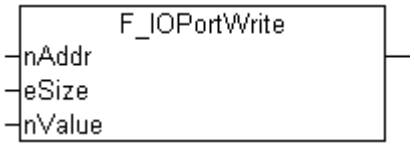
nAddr: I/O Portadresse.

eSize: Anzahl [▶ 112] der zu lesenden Datenbytes.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcSystem.Lib

4.2.2 F_IOPortWrite



Mit der Funktion F_IOPortWrite kann eine eSize-breite I/O Position beschrieben werden. Siehe auch die Beschreibung der F_IOPortRead [▶ 91] Funktion.

Ein direkter Hardwarezugriff stellt keine Probleme dar, solange Daten nur gelesen werden.

HINWEIS

Absturz durch Schreibzugriff

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.

BITTE BENUTZEN SIE DIESE FUNKTION NUR, WENN SIE GENAU WISSEN, WAS SIE TUN!

FUNCTION F_IOPortWrite : BOOL

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

nAddr: I/O Portadresse.

eSize: Anzahl [▶ 112] der Datenbytes die geschrieben werden sollen.

nValue: 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
TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcSystem.Lib

4.3 Ads Funktionen

4.3.1 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 (4Byte 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

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

msgCtrlMask : Kontrollmaske, die den Typ und die Wirkung der Meldungsausgabe bestimmt (siehe separate Tabelle).

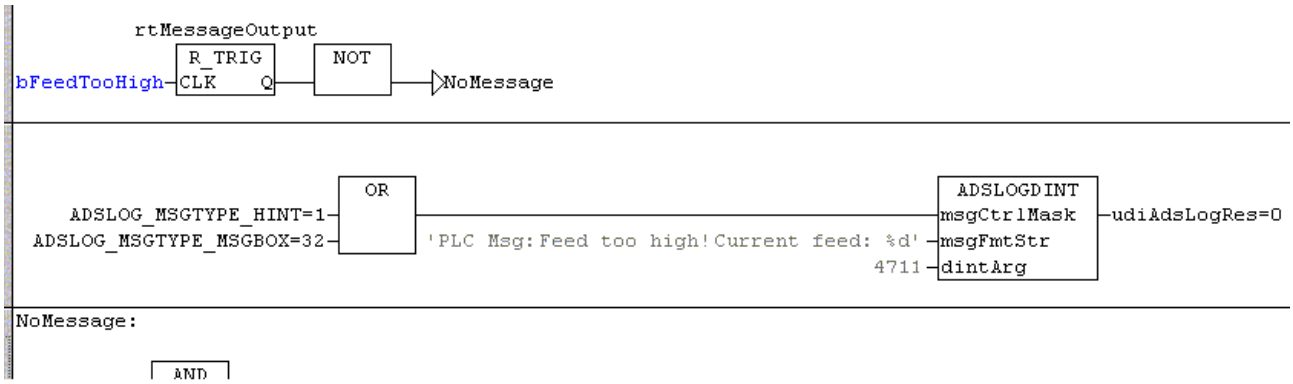
Konstante	Beschreibung
ADSLOG_MSGTYPE_HINT	Meldungstyp ist Hinweis.
ADSLOG_MSGTYPE_WARN	Meldungstyp ist Warnung.
ADSLOG_MSGTYPE_ERROR	Meldungstyp ist Fehler.
ADSLOG_MSGTYPE_LOG	Meldung wird in das Logbuch geschrieben.
ADSLOG_MSGTYPE_MSGBOX	Meldung wird in einer Messagebox ausgegeben.
ADSLOG_MSGTYPE_RESOURCE	Meldung wird aus einer Ressource-Datei geholt. (wird momentan nicht unterstützt)
ADSLOG_MSGTYPE_STRING	Meldung ist direkt angegebener String (default).

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

msgFmtStr : Enthält die auszugebende Meldung [▶ 109]. Sie kann das Formatierzeichen ‚%d' für die Ausgabe eines DINT-Wertes an beliebiger Stelle enthalten.

dintArg : Enthält den in die Meldung einzufügenden numerischen Wert .

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:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.2 ADSLOGLREAL

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 auszugebende 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 bitte, dass auch hier, wie im Beispiel gezeigt, die Funktion flankengesteuert aufgerufen werden muss (siehe auch Beschreibung ADSLOGDINT [▶ 94]). Der Rückgabeparameter enthält den Funktionsfehlercode oder 0 falls erfolgreich.

FUNCTION ADSLOGLREAL : DINT

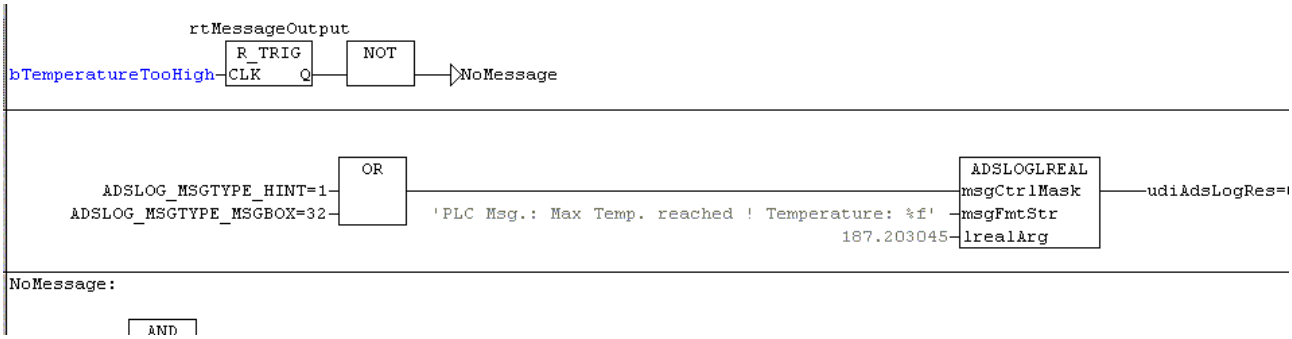
```
VAR_INPUT
    msgCtrlMask     : DWORD;
    msgFmtStr       : T_MaxString;
    lrealArg        : LREAL;
END_VAR
```

msgCtrlMask : Kontrollmaske, die den Typ und die Wirkung der Meldungsausgabe bestimmt (siehe separate Tabelle). Aufzählung aller z. Zt. in der Bibliothek als globale Konstanten implementierten Kontrollmasken für die Meldungsausgabe (s. Beschreibung der Funktion ADSLOGDINT [▶ 94]).

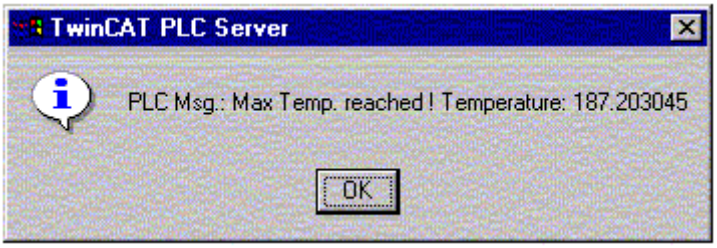
msgFmtStr : Enthält die auszugebende Meldung [▶ 109]. Sie kann das Formatierzeichen ‚%d' für die Ausgabe eines DINT-Wertes an beliebiger Stelle enthalten.

lrealArg : Enthält den in die Meldung einzufügenden numerischen Wert .

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:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.3 ADSLOGSTR

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 auszugebende 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.

Hier muss, wie auch im Beispiel gezeigt, die Funktion flankengesteuert aufgerufen werden (siehe auch Beschreibung ADSLOGDINT [▶ 94]).

Der Rückgabeparameter enthält den Funktionsfehlercode oder 0 falls erfolgreich.

FUNCTION ADSLOGSTR : DINT

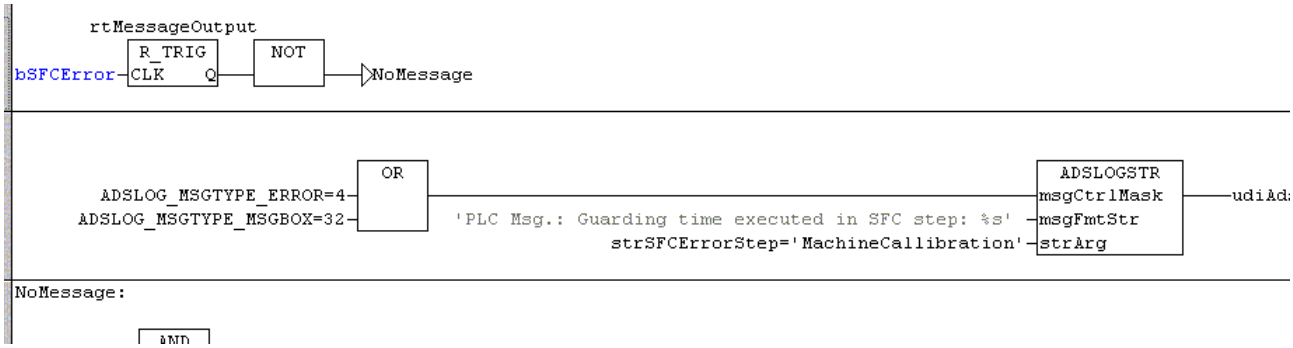
```
VAR_INPUT
    msgCtrlMask : DWORD;
    msgFmtStr   : T_MaxString;
    strArg      : T_MaxString;
END_VAR
```

msgCtrlMask : Kontrollmaske, die den Typ und die Wirkung der Meldungsausgabe bestimmt (siehe separate Tabelle bei ADSLOGDINT [▶ 94]).

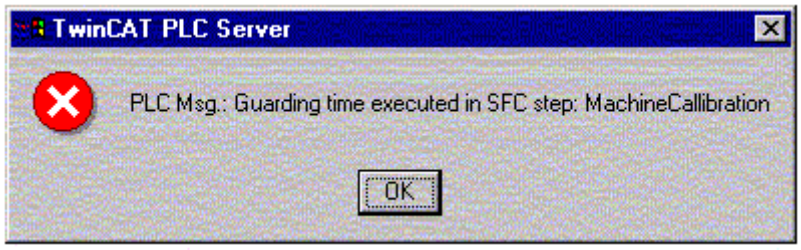
msgFmtStr : Enthält die auszugebende Meldung [▶ 109]. Sie kann das Formatierzeichen ‚%d' für die Ausgabe eines DINT-Wertes an beliebiger Stelle enthalten.

strArg : Enthält den in die Meldung einzufügenden String .

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.

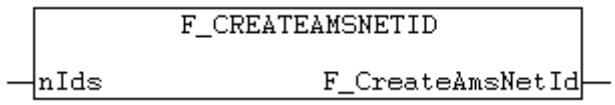
Beispiel für den Aufruf der Funktion in ST:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.3.4 F_CreateAmsNetId



Die Funktion generiert einen formatierten NetId-String und liefert diesen als Rückgabeparameter zurück (z.B.: '127.16.17.3.1.1').

FUNCTION F_CreateAmsNetId : T_AmsNetId

T_AmsNetId [▶ 107]

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

nIds : Byte-Array [▶ 108]. Jedes Byte entspricht einer Nummer der Netzwerkadresse. Die Adressbytes haben eine Netzwerk-Byte-Reihenfolge.

Beispiel für einen Aufruf in ST:

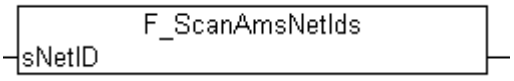
```
PROGRAM MAIN
VAR
    ids     : T_AmsNetIdArr := [127, 16, 17, 3, 1, 1];
    sNetID  : T_AmsNetID := '';
END_VAR
```

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

Voraussetzungen

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

4.3.5 F_ScanAmsNetIds



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. Die Adressbytes haben eine Netzwerk-Byte-Reihenfolge.

FUNCTION F_ScanAmsNetIds : T_AmsNetIdArr

T_AmsNetIdArr [▶ 108]

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

sNetID: TwinCAT Netzwerkadresse [▶ 107] als String. 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
```

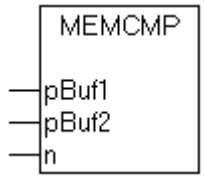
```
ids := F_ScanAmsNetIds( sNetID );
(* Result: ids[0]:=127, ids[1]:=16, ids[2]:=17, ids[3]:=3, ids[4]:=1, ids[5]:=1 *)
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1257	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4 MEMORY Funktionen

4.4.1 MEMCMP



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

FUNCTION MEMCMP : DINT

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

pBuf1: Anfangsadresse des ersten Speicherbereichs (der erste Datenpuffer).

pBuf2: Anfangsadresse des zweiten Speicherbereichs (der zweite Datenpuffer).

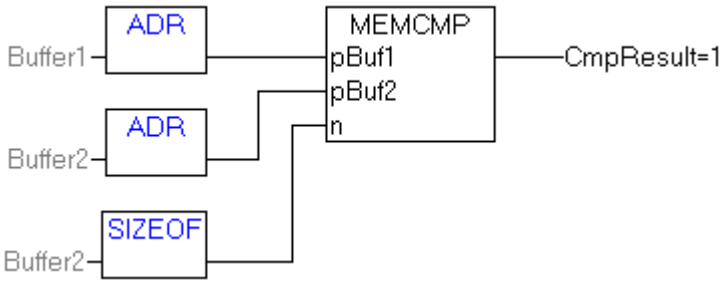
n: Anzahl der zu vergleichenden Bytes.

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

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

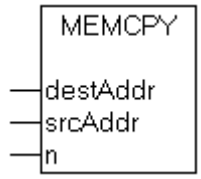


Im Beispiel werden 4 Byte Daten vom *Buffer2* mit dem *Buffer1* verglichen. Das erste unterschiedliche Datenbyte ist im *Buffer1* größer als in *Buffer2*.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCHelper.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4.2 MEMCPY



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

FUNCTION MEMCPY : UDINT

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

destAddr: Anfangsadresse des Ziel-Speicherbereichs.

srcAddr: Anfangsadresse des Quell-Speicherbereichs.

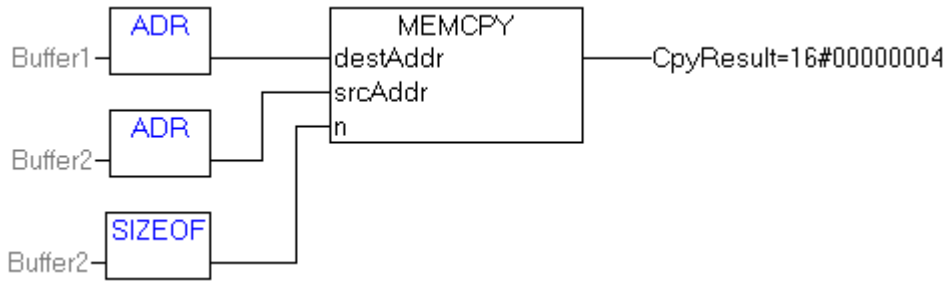
n: Anzahl der zu kopierenden Bytes.

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

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

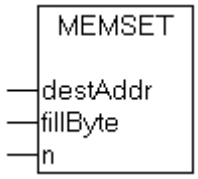


Im Beispiel werden 4 Byte vom *Buffer2* nach *Buffer1* kopiert.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCHelper.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4.3 MEMSET



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

FUNCTION MEMSET : UDINT

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

destAddr: Anfangsadresse des zu setzenden Speicherbereichs.

fillByte: Wert der Füll-Bytes.

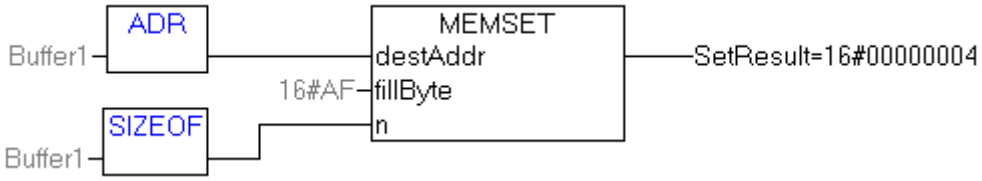
n: Anzahl der zu setzenden Bytes.

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

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

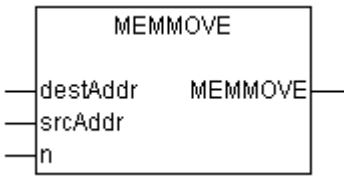


Im Beispiel werden 4 Byte im *Buffer1* auf den Wert 0xAF gesetzt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCHelper.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.4.4 MEMMOVE



Mit der Funktion MEMMOVE können Werte der SPS-Variablen von überlappenden Speicherbereichen kopiert werden.

FUNCTION MEMMOVE : UDINT

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

destAddr: Anfangsadresse des Ziel-Speicherbereichs.

srcAddr: Anfangsadresse des Quell-Speicherbereichs.

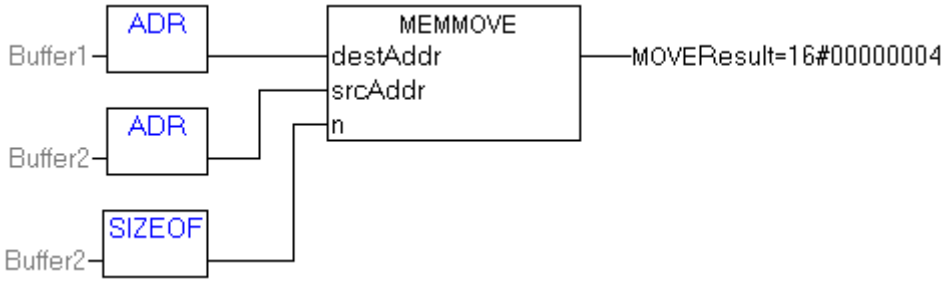
n: Anzahl der zu kopierenden Bytes.

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:

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



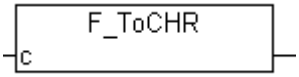
Im Beispiel werden 4 Byte vom *Buffer2* nach *Buffer1* verschoben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.8.0 Build > 737	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.5 Character Funktionen

4.5.1 F_ToCHR



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

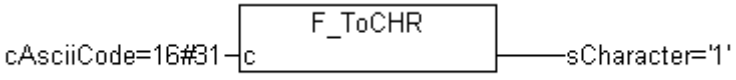
FUNCTION F_ToCHR: STRING

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

c: Der zu konvertierende ASCII-Code;

Beispiel für den Aufruf der Funktion in FBD:

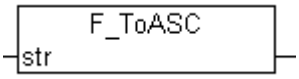
```
PROGRAM P_TEST
VAR
    sCharacter : STRING(1) := '';
    cAsciiCode : BYTE := 16#31;
END_VAR
```



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.10.0 Build > 1256	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

4.5.2 F_ToASC



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.

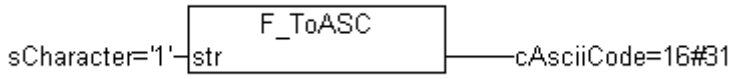
FUNCTION F_ToASC : BYTE

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

str: Der zu konvertierende 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
TwinCAT v2.10.0 Build > 1256	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5 Datentypen

5.1 SYSTEMINFOTYPE

```
TYPE SYSTEMINFOTYPE
STRUCT
    runTimeNo           : BYTE;
    projectName         : STRING(32);
    numberOfTasks       : BYTE;
    onlineChangeCount   : UINT;
    bootDataFlags       : BYTE;
    systemStateFlags    : WORD;
END_STRUCT
END_TYPE
```

runTimeNo : Gibt die Nummer des Laufzeitsystems (1..4) an.

projectName : Name des Projekts als STRING.

numberOfTasks : Anzahl der im Laufzeitsystem befindlichen Tasks (max. 4).

onlineChangeCount : Anzahl der seit dem letzten Komplettdownload gemachten Online-Änderungen.

bootDataFlags : Zustand der Bootdaten (RETAIN und PERSISTENT) nach dem Laden. Die oberen vier Bits signalisieren den Zustand der persistenten Daten, die unteren vier Bits den Zustand der Retain-Daten.

Bitnummer	Beschreibung
0	RETAIN Variablen: LOADED (fehlerfrei geladen)
1	RETAIN Variablen: INVALID (es wurde die Sicherungskopie geladen, weil keine gültige Datei vorhanden war)
2	RETAIN Variablen: REQUESTED (RETAIN Variablen sollten geladen werden, Einstellung im TwinCAT System Control))
3	reserviert
4	PERSISTENT Variablen: LOADED (fehlerfrei geladen)
5	PERSISTENT Variablen: INVALID (es wurde die Sicherungskopie geladen, weil keine gültige Datei vorhanden war)
6	reserviert
7	reserviert

systemStateFlags : Reserviert.

Beim Shutdown (Stopp) von TwinCAT werden die PERSISTENT und die RETAIN Daten in zwei Dateien auf die Festplatte geschrieben. Der Pfad kann im TwinCAT System Control über die TwinCAT System Eigenschaften (Reiter PLC) angegeben werden. Die Standardeinstellung ist "*<Laufwerk>:\TwinCAT\Boot*". Die Dateien haben alle einen festen Namen und eine feste Endung.

Dateiname	Beschreibung
TCPLC_P_x.wbp	Bootprojekt (x = Nummer des Laufzeitsystems)
TCPLC_S_x.wbp	Gepackter Sourcecode (x = Nummer des Laufzeitsystems)
TCPLC_R_x.wbp	RETAIN Variablen (x = Nummer des Laufzeitsystems)
TCPLC_T_x.wbp	PERSISTENT Variablen (x = Nummer des Laufzeitsystems)
TCPLC_R_x.wb~	Sicherungskopie der RETAIN Variablen (x = Nummer des Laufzeitsystems)
TCPLC_T_x.wb~	Sicherungskopie der PERSISTENT Variablen (x = Nummer des Laufzeitsystems)

Kann beim Shutdown (Stopp) von TwinCAT die Datei der persistenten und/oder retain Variablen nicht geschrieben werden, so wird standardmäßig die Sicherungsdatei geladen. In der SPS ist dann im bootDataFlags das Bit 1 (für die RETAIN Variablen) oder/und das Bit 5 (für die PERSISTENT Variablen) gesetzt.

Soll die Sicherungsdatei auf keinen Fall verwendet werden, so ist in der NT Registry eine Einstellung vorzunehmen. Hierzu ist mit dem Registry-Editor unter

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\Plc]

"ClearInvalidRetainData"=dword:00000000

"ClearInvalidPersistentData"=dword:00000000
```

der Wert von "ClearInvalidRetainData" auf 1 bzw. von "ClearInvalidPersistentData" auf 1 zu setzen. Die Defaulteinstellung ist 0.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.2 SYSTEMTASKINFOTYPE

```
TYPE SYSTEMTASKINFOTYPE
STRUCT
    active              : BOOL;
    taskName            : STRING(16);
    firstCycle          : BOOL;
    cycleTimeExceeded   : BOOL;
    cycleTime           : UDINT;
    lastExecTime        : UDINT;
    priority            : BYTE;
    cycleCount          : UDINT;
END_STRUCT
END_TYPE
```

active : Diese Variable zeigt an, ob die Task aktiv ist.

taskName : Der Taskname.

firstCycle : Diese Variable hat im ersten Zyklus der SPS-Task den Wert: TRUE.

cycleTimeExceeded : In dieser Variablen wird ein Überschreiten der eingestellten Taskzykluszeit gemeldet.

cycleTime : Eingestellte Taskzykluszeit in Vielfachen von 100ns.

lastExecTime : Benötigte Zykluszeit für den letzten Zyklus in Vielfachen von 100 ns.

priority : Eingestellte Priorität der Task.

cycleCount : Zykluszähler.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	PLCSystem.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.3 T_AmsNetId

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

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. Jedes Zahlenfeld enthält eine Zahl zwischen 0 und 254. 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.

Voraussetzungen

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

5.4 T_AmsNetIdArr

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

Eine SPS-Variable von diesem Typ enthält die einzelnen Adressbytes der AMS-Netzwerkkennung. 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: F_ScanAmsNetIds [▶ 99]

Voraussetzungen

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

5.5 T_AmsPort

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

ADS-Geräte im TwinCAT-Netzverbund werden durch eine AMS-Netzwerkadresse und eine Portnummer identifiziert. Auf jedem TwinCAT-Einzelsystem sind folgende dezimale Portnummern invariant festgelegt.

Tabelle mit festgelegten ADS-Portnummern:

ADS-Gerät	Portnummer
Nockenschaltwerk	900
Laufzeitsystem 1 Laufzeitsystem 2 Laufzeitsystem 3 Laufzeitsystem 4	801 811 821 831
NC	500
Reserviert	400
I/O	300
Echtzeitkern	200
Meldesystem (Logger)	100

Es können bis zu 4 unabhängige SPS-Laufzeitsysteme (LZS) auf einem Rechner laufen, jedes LZS ist als eigenständige SPS zu betrachten. Die Portnummer wird, zusammen mit der Netzwerkadresse, beim Aufruf der ADS-Bausteine als Eingangsparameter benötigt.

Voraussetzungen

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

5.6 T_MaxString

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

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.

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

Voraussetzungen

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

5.7 T_IPv4Addr

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

String containing an (Ipv4) Internet Protocol dotted address. E.g. '172.16.7.199'

Voraussetzungen

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

5.8 T_IPv4AddrArr

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

Eine SPS-Variable von diesem Typ enthält die einzelnen Adressbytes der (IPv4) Internet Protocol-Netzwerkadresse. 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: F_ScanIPv4AddrIds [▶ 84]

Voraussetzungen

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

5.9 ST_AmsAddr

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

netId: Ams network ID address [▶ 108] bytes;

port: Ams port number [▶ 108];

Voraussetzungen

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

5.10 E_OpenPath

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

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

Voraussetzungen

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

5.11 E_SeekOrigin

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

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

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
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.12 E_TcEventClass

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.13 E_TcEventClearModes

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.14 E_TcEventPriority

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.15 E_TcEventStreamType

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.16 E_IOAccessSize

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

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.0.10 Build > 1257	PC or CX (x86)	TcSystem.Lib

5.17 TcEvent

```
TYPE TcEvent
STRUCT
    Class               : UDINT;
    Prio                : UDINT;
```

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

Class : Eventklasse, Wert aus dem Enum E_TcEventClass [▶ 111] entnehmen

Prio : Priorität des Events innerhalb einer Klasse, frei wählbare Zahl (1..MaxUDINT)

Id : Id des Events, wird zur eindeutigen Identifizierung im Eventlogger genutzt

bQuitRequired : Flag zum Ein- und Ausschalten der Quittierpflichtigkeit (TRUE --> Quittierpflichtig)

DataFormatStrAddress : Adresse eines Strings, String enthält Formatieranweisungen (z.B. '%d%f' formatiert einen Integer und einen Real (float) Wert)

UserFlags : 32-bit Zahl zur freien Verfügung

Flags : 32-bit Zahl zur Kennzeichnung des Events, die Bedeutung der einzelnen Bits sind in den globalen Variablen [▶ 115] der Bibliothek deklariert

StreamType : Typ des Events, Wert aus dem Enum E_TcEventStreamType [▶ 112] entnehmen.

SourceString : String mit dem Sourcenamen (max. 15 Zeichen [▶ 115]).

SourceId : Source-ID.

ProgId : String (Prog-Id) mit dem Namen des Formatters (max. 31 Zeichen [▶ 115])

TwinCAT 2.7 default: 'TcEventLogger.TcLogFormatter'

TwinCAT 2.8 default: 'TcEventLogger.TcLogFormatter' oder 'TcEventFormatter.TcXmlFormatter'

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.7.0	PC or CX (x86)	Standard.Lib, PLCEvent.Lib, PLCSystem.Lib, TcPlcAds.Lib
TwinCAT v2.8.0	PC or CX (x86)	TcSystem.Lib
TwinCAT v2.10.0 Build >= 1301	CX (ARM)	TcSystem.Lib

5.18 PVOID

```
TYPE PVOID : UDINT;
END_TYPE
```

Platform independent pointer type:

- x86, ARM => 32 bit pointer type (TwinCAT 2.xx default);
- x64 => 64 bit pointer type (reserved for future use in TwinCAT 3.x);

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

5.19 UXINT

```
TYPE UXINT : UDINT;
END_TYPE
```

Platform independent unsigned integer type:

- x86, ARM => 32 bit unsigned integer (TwinCAT 2.xx default);
- x64 => 64 bit unsigned integer (reserved for future use in TwinCAT 3.x);

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

5.20 XINT

```
TYPE XINT : DINT;
END_TYPE
```

Platform independent signed integer type:

- x86, ARM => 32 bit signed integer (TwinCAT 2.xx default);
- x64 => 64 bit signed integer (reserved for future use in TwinCAT 3.x);

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

5.21 XWORD

```
TYPE XWORD : DWORD;
END_TYPE
```

Platform independent bit type:

- x86, ARM => 32 bit unsigned integer (TwinCAT 2.xx default);
- x64 => 64 bit unsigned integer (reserved for future use in TwinCAT 3.x);

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 2243	PC or CX (x86, ARM)	TcSystem.Lib

6 Konstanten

In der TcSystem.lib werden verschiedene Konstanten definiert.

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 PLC Servers in der Runtime 1.
AMSPORT_R0_PLC_RTS2	811	Portnummer des TwinCAT PLC Servers in der Runtime 2
AMSPORT_R0_PLC_RTS3	821	Portnummer des TwinCAT PLC Servers in der Runtime 3
AMSPORT_R0_PLC_RTS4	831	Portnummer des TwinCAT 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	27110	Portnummer des TwinCAT Scope Servers (ab Lib. V2.0.12)

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
ADSSTATE_RECONFIG	16	ADS Status: Reconfiguration
ADSSTATE_MAXSTATES	17	

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 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	

System Service Index Groups	Wert	Beschreibung
SYSTEMSERVICE_FDELETE	131	
SYSTEMSERVICE_FRENAME	132	
SYSTEMSERVICE_REG_HKEYLOCALMACHINE	200	
SYSTEMSERVICE_SENDEMAIL	300	
SYSTEMSERVICE_TIMESERVICES	400	
SYSTEMSERVICE_STARTPROCESS	500	
SYSTEMSERVICE_CHANGENETID	600	

System Service Index Offsets (Timeservices)	Wert	Beschreibung
TIMESERVICE_DATEANDTIME	1	
TIMESERVICE_SYSTEMTIMES	2	
TIMESERVICE_RTCTIMEDIFF	3	
TIMESERVICE_ADJUSTTIMETORTC	4	

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	

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

Masken für BSOD-Flags	Wert	Beschreibung
SYSTEMSTATEFLAGS_RTVIOLATION	16#02	Echtzeit-Verletzung, Latenzzeitüberschreitung

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.

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 Status-meldungen	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	The max. string length of T_MaxString data type

Mehr Informationen:
www.beckhoff.de/tx1200

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de